FPGA とデータセンター: スタックの必要性

この記事は、インテル® デベロッパー・ゾーンに公開されている「FPGAs and Data Centers: It Takes a Stack」の日本語参考訳です。

信号処理からネットワーク・パケット処理、暗号化、ディープラーニングの推論に至るまで、広範なアプリケーションの設計から、FPGA はアルゴリズムの実行においてパフォーマンスと電力消費を大幅に向上することが分かっています。一般にこの向上は、FPGA ハードウェアで計算カーネル (アルゴリズムの内部ループ) を実装し、これらのカーネルを CPU ソフトウェアからオフロードして、実行に大規模な並列処理や深いパイプラインを適用することでもたらされます。このことから、人工知能、ストリーミング・データ解析、ネットワーク機能の仮想化、従来のスーパーコンピューティング・アプリケーションを含む、計算集約型のワークロードが増加しているクラウドとエンタープライズ・データセンターにおいて、FPGA はデータセンターとワークロードの双方に大きなメリットをもたらすと考えられます。実際、大手パブリック・クラウド・プロバイダーは、一部のワークロード向けに FPGA アクセラレーションの提供を開始しました。数百万ものレジスターと数百メガビットもの内部メモリーを備えた最新世代の FPGA が市場に浸透するにつれて、この傾向は確実に加速するでしょう。

しかし実際には、大規模 FPGA の経験豊富なユーザーはすでに知っているように、従来の FPGA 開発手法とデータセンターのソフトウェア中心で高レベルの文化 (アプリケーション開発者、DevOps マネージャー、およびワークロード管理の自動化プロセス) の間には大きな隔たりがあります (図 1)。この違いは、プログラミングと MANO (管理とオーケストレーション) の 2 つの点で最も深刻です。

図 1. 従来の FPGA 開発者とデータセンター開発者の文化の間には隔たりがある

プログラミングとは?

最初の違いは、「プログラミング」という用語が CPU と FPGA では異なる意味を持つことが原因です。それぞれにおいて「プログラミング」は、異なる種類のタスクを指し、異なるスキルを意味します。

CPU では、データ構造を定義して、CPU が実行する命令の順序付けされたリストを作成することを意味します。CPU ハードウェアはブラックボックスであり、プログラミング・ツール・チェーンによって生成された機械言語命令を正しく実行します。セマフォーなどによって順序付けに制約を課すことはできますが、プログラマーからは命令実行の実際の順序とタイミングは見えません。

一方、FPGA プログラミングも一般に人間が解読可能な言語と翻訳ツールから開始します。しかし、高度なツールや C++ などの高水準言語は、命令シーケンスを作成するためではなく、アルゴリズムを記述するために使用されます。この 2 つの目的では、プログラミング・スタイルとベスト・プラクティスが大きく異なります。アルゴリズムを記述するため、ツールチェーンは次の新たな段階に進みます。これは、プログラミング言語のように見えるかもしれませんが、実際にはハードウェア構造を指定する、Verilog* などのハードウェア記述言語 (HDL)

と呼ばれるものです。この段階では、ビットとバイトでデータ構造が定義されます。実行可能な命令ではなく、ロジックのブロックを定義してレジスターに関連付け、要素間の相互接続を定義する HDL コードだけです。この時点では、言語ツールは FPGA ロジックの構成を指定しているため、この開発プロセスはプログラミングと言うよりも「構成」と言ったほうが適切かもしれません。

そして、FPGA ツールチェーンは、ソフトウェア開発にはない次の段階に進みます。最初に、FPGA 開発者は後続のステップを管理するタイミング制約を指定する必要があります。次に、自動化ツールがタイミング制約を満たすように、設計の HDL 記述をロジックとストレージノードのネットワーク、そして FPGA の実際のハードウェア・マクロ関数、ロジックエレメント、インターコネクト・セグメントにマップします。FPGA 開発者は、このチェーン全体をプログラミングと呼ぶかもしれません。実際にプログラミングのようなものから始まりますが、すぐにハードウェアの構成という非常に異なるものになります。

MANO

データセンターと FPGA を分け隔てているもう 1 つの文化的な違いは、複雑で難解な自動化された管理とオーケストレーションです。数千ものアプリケーションがスタンバイ状態とアクティブ状態の間を遷移しているクラウド・データセンターでは、場合によってはミリ秒で自動化を制御する必要があります。リソースを追跡、更新、保守して、請求する必要があります。ワークロードを認証し、リソース要求を満たし、実行を監視して、正常に終了する必要があります。これらすべてをデータセンターのインフラストラクチャーを最適に使用して行う必要があります。これを実現するため、MANO ツールはデータセンター・ファブリックの均一性に依存しています。また、データセンターのリソースをネットワークを介してサーバーに仮想的に供給できることも重要です。

控えめに言っても、FPGA がこの環境にどのように適合するかは明らかではありません。FPGA は固定ではないため、データセンター内の各 FPGA は、理論上は異なる構成を持つことが可能です。それらの構成はミリ秒単位では変更できますが、ナノ秒では変更できません。そして、それらはファブリックのすべての計算ノードに存在するわけではありません。MANO は均一性を求め、FPGA は機能、空間、時間に多様性をもたらします。

不一致の解消

特定の機能 (仮想レイヤー 2 スイッチング・デバイスや暗号化アクセラレーターなど) を実行するため FPGA をデータセンター・ファブリックに追加すると、チップを特定の機能に固定し、ユーザーに対して透過的に保つことで、不一致の問題を回避できます。しかし、FPGA の価値をアプリケーション開発者とエンドユーザーが利用できるようにするには、単にチップをカードに配置してカードをサーバースロットに装着するだけでなく、さらに多くの作業が必要になります。この取り組みはハードウェア・レベルから開始し、抽象化レベルを上げながら、ソフトウェア開発者やデータセンター・オペレーターのレベルに至るまでの段階を経ていく必要があります。この作業は、ネットワーク・エンジニアの概念を借りて、スタックとしてモデル化できます。ここでは、物理、構成、抽象化、環境、MANO の 5 つのレイヤーを使用します (図 2)。

図 2. FPGA アクセラレーションをユーザーが利用できるようにするために必要なタスクとツールを概念的に示した適合スタック

FPGA シリコンのレベルでは、設計者は数々の決定を下す必要があります。単純にサーバーボード上の CPU ソケットに差し込む FPGA チップキャリアを作成することは、温度、電力、機械的な要因から技術的に可能ですが、現実的ではありません。さまざまなことを考慮すると、データセンターのサーバーラックに装着する専用カードに FPGA を配置して、チップのシリアルポートがラックのバックプレーン・ネットワークにアクセスできるようにするアプローチが考えられます。FPGA とその高速なインパッケージ・メモリーを特定の CPU と密接に結合するため、カードは PCI Express* (PCIe*) を介してサーバー CPU にブリッジできます。この方法では、FPGA は CPU に対するスレーブ・アクセラレーターとして動作することも、ネットワークから直接データをストリーミングすることもできます。このアプローチは、例えば、インテル® Stratix® 10 プログラマブル・アクセラレーター・カードで採用されています。

このレベルでは、それほど明白ではないもう 1 つの設計努力が必要とされます。プログラムされていないと、FPGA は PCIe* インターフェイスへの応答と認証、復号化、構成ファイルのロード以外、ほとんど何もできません。ホストとの通信、セキュリティーの維持、データ転送、実行の管理など、FPGA を便利なアクセラレーターとする機能をチップに設定する必要があります。これらの機能は、シグナルブリッジと呼ばれる、ユーザー定義関数が利用可能なハードウェア・ガスケットを作成します。ガスケットの設計には、FPGA、CPU、サーバーボード設計者の協力が必要なことは明らかです。

これらの物理条件は、FPGA でアクセラレーション関数を初期化し、制御することを可能にします。さらに、オペレーティング・システムのカーネルと統合されたデバイスドライバーは、OS、ハイパーバイザー、ユーザー・ワークロードに対してこれらの制御と監視操作を物理的に利用可能にします。ただし、利用可能であっても、実際に利用できるとは限りません。スタックにはこのほかのレイヤーもあります。

構成

高度に抽象的なものからハードウェアへの依存性が高いものまで、さまざまなレベルのプログラミング言語があるように、FPGA プログラミングと構成ツールにもいくつかの種類があり、それぞれ異なるエントリーポイントを FPGA 開発者に提供します。C++ プログラムを FPGA 構成にほぼ直接マップできるツールから、開発者が HDL で直接構成できるようにするツールまで、さまざまなものがあります。チップ内の個々の構成ビットを操作する低レベルのツールもありますが、それらは一般に FPGA ベンダーによってのみ使用されます。

ハードウェア・スキルを持ち合わせていないアルゴリズム開発者向けに、ベンダーのネイティブ・ツール・チェーンで使用するため、ソフトウェア (通常は C/C++ 言語) を HDL コードに変換するツールがあります。これらのツールは大きく 2 つに分けることができます。FPGA が主に組込みシステムで小さな機能のアクセラレーションに使用されていた時代からあるツールでは、開発者が C/C++ のサブセットでロジックブロックの動作を記述し、ツールがそのブロックの HDL コードを生成します。生成されたコードは、C、I/O 関数、前述のガスケットでは記述することが困難な関数とリンクされ、完全なアクセラレーターの記述が生成されます。そして、この HDL はベンダーツールに供給され、合成、検証、タイミングチェック、FPGA ハードウェアへのマップが行われます。最後に、ベンダーツールによって、チップ上の個々の構成レジスターへロードする値を含む構成ファイルが作成されます。このプロセスは、アクラレーター・ブロックの開発期間を短縮できますが、設計がチェーンのすべての段階を経るためには、FPGA 開発者の経験が必要になる場合があります。そしてそれは、FPGA の膨大なリソースをアルゴリズムに利用するため、並列プログラミングやパイプライン構造に関する経験を前提としています。

ごく最近では、並列処理、パイプライン、ストリーミングの概念を理解するためのやや高レベルのツールに強い関心が寄せられています。業界標準の OpenCL* 言語では、開発者はアルゴリズムを、ホスト CPU で実行する C で記述されたメインまたは制御ブロックと、高速化する 1 つ以上 のカーネルの 2 つに分割します。メインブロックは、データを移動し、並列処理を考慮したアプリケーション・プログラミング・インターフェイス (API) を利用してカーネルの実行を制御します。

カーネルは、プラグマと C サブセットでその並列化と管理方法が記述されています。コンパイラー・ディレクティブを使用して、開発者はデバッグのためすべて CPU 上で実行するようにコードをコンパイルするか、またはメインボディーは CPU 上で実行し、カーネルは指定した FPGA リソースで並列化するようにコードをコンパイルします。後者では、カーネルコードは HDL に変換され、自動化プロセスがそれを合成し、適切なタイミング制約を自動生成し、ネットリストを FPGA 領域にマップして、それをガスケットと必要な I/O にリンクします。バックエンド・タスクの自動化は、理論上の最大パフォーマンスをわずかに低下させますが、FPGA エキスパートによる介入の必要性をほとんど排除し、展開までの時間を大幅に短縮します。

アクセラレーションの可能性を特定し、カーネルに実装するプロセスは、簡単でもなければ、常に直感的でもありません。インテル® Parallel Studio XE など、並列化と調査を支援するツールは、通常 CPU のベクトル・プログラミング向けに開発されたものであり、FPGA 上で並列ハードウェアを作成するためのものではありませんが、非常に役に立ちます。

抽象化

FPGA ツールではいくつかの抽象化レベルを利用できますが、多くのアプリケーション開発者は、並列プログラミングと FPGA 開発のささいな違いであっても、彼らの目的には無関係であると見なします。彼らに必要なものは、さらなる自動化ツールではなく、FPGA を設計し、アクセラレーターを初期化して特定の機能の FPGA アクセラレーションを呼び出すライブラリーを提供してくれる外部の専門家です。

これはすでにアプリケーション全般で広く使用されているライブラリーでは行われています。クラウドでの FPGA の導入が進むにつれて、より専門的なライブラリーでもこのアプローチが採用されるでしょう。これらは、構造解析や流体解析、物理化学、遺伝学など、特定の分野の取り組みを対象とします。

ライブラリーの次のステップは、マシンラーニング、データ分析、ビデオコーディングなどのアプリケーション・フレームワークへの FPGA アクセラレーションの統合です。これもすでに行われています。このレベルでは、ユーザーは FPGA のアーキテクチャーと動作の問題から完全に除外されています。ユーザーは FPGA を意識しておらず、パフォーマンスを大幅に向上する設定としか認識していないでしょう。

この傾向が拡大するにつれて、データセンターから FPGA アクセラレーションを要求し、利用可能な場合は適用するターンキー・アプリケーションが増加するでしょう。現時点では、アルゴリズムの適応、ライブラリー開発、テスト、統合に多くの労力が費やされていますが、アプリケーション・ユーザーには実行時間の大幅な短縮や高スループットしか見えていません。

開発環境

テストの問題は、スタックの次のレベルである環境レイヤーにつながります。FPGA 実装の開発方法に関係なく、ほかのクラウド・アプリケーションと同じデバッグ環境が必要になります。DevOps がパッケージの場所、分離、実行、自動展開を管理します。しかし、FPGA アクセラレーション・パッケージには、ソフトウェアのみのワークロードには存在しない重大な問題があります。

FPGA を考慮した環境は、チップに構成ファイルをロードし、FPGA メモリーを読み書きし、FPGA ロジックを初期化し、実行を開始/停止するため、チップのドライバーにアクセスできなければなりません。これらの操作のほとんどは、サーバー CPU 上のドライバーコードに加えて、FPGA のガスケットロジックによるサポートを必要とします。FPGA アクセラレーション・コードのデバッグは、ソフトウェアのみのデバッグとは異なります。FPGA には、CPU のトレース、シングルステップ、ブレークポイント機能と同等のものはなく、printf と同等の機能もありません。これらの機能のほとんどは、ガスケットロジックからのサポートとユーザーのロジック設計にデバッグ構造を明示的に含めることを必要とします。

FPGA アクセラレーション・コードのデバッグは、直感と推測で行われるわけではありません。アプリケーション開発者に有用な FPGA 構成のソフトウェア・シミュレーションを提供するツールがあります。OpenCL* 向けのツールでは、CPU コードまたはカーネルの HDL を生成し、開発者がまずソフトウェアのみの環境で高レベルのデバッグを行うことができます。

さらに深いレベルでは、経験豊富な FPGA 開発者が個々の FPGA レジスターを視覚的に確認できる SignalTap ロジック・アナライザーなどのツールもあります。しかし、検証済みのライブラリーとフレームワークを使用するほとんどのユーザーは、そのようなツールを必要とせず、コンパイル済みのソフトウェア関数を呼び出すのと同じように、ソースレベルでコードをデバッグします。

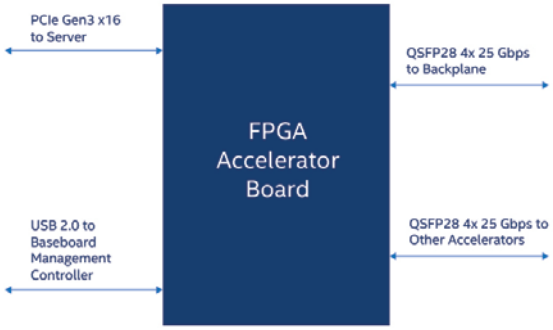
MANO

クラウド・データセンターの多くのタスクは、ユーザーコードを直接実行しません。管理タスクは、ファブリック内のどこにリソースがあるのか追跡し、更新が完全に展開されていることを確認し、請求のためリソースの使用を追跡し、例外に対応します。オーケストレーション・タスクは、理想的にはリソース使用率とタスク・パフォーマンスの両方を最適化する方法で、計算、ネットワーク、ストレージリソースをタスクに割り当てます。ハイパーバイザーは、仮想マシンを作成し、それらを互いに保護し、物理リソースにバインドします。これらのタスクはすべて、FPGA アクセラレーターの特殊な状況に対処する必要があります。

問題は、CPU とは異なり、FPGA にはユーザーレベルの機能がありません。FPGA は、計算リソースをインストールできるレセプタクルのようなものです。管理自動化ソフトウェアの場合、FPGA のドライバーと可能であればインテルの Open Programmable Accelerator Engine などのオープンスタンダード・インターフェイスを介して、FPGA にあるガスケットコードのバージョン、実装されているユーザー機能、ほかの機能に利用可能なリソースを調査することを意味します。また、ドライバーからの例外信号を解釈して、適切に処理することも意味します。

オーケストレーションの自動化の課題はやや異なります。各 FPGA にプログラムされているリソース、コミットされていないリソース、データセンター・ファブリックへのチップの接続方法 (サーバーは PCI Express* (PCIe*)、データセンター・ネットワークはイーサネット、その他の FPGA はプライベート・リンク、 あるいはそれらの組み合わせ) を理解する必要があります (図 3)。FPGA の内部メモリーの割り当ても理解する必要があります。また、後で使用するため FPGA 構成を保持する場合と、削除してその一部またはすべてを再構成する場合 (構成には時間がかかります) を決定しなければなりません。この問題は、粒度の細かい不揮発性メモリーの大きなブロックを持つサーバーのオーケストレーションで直面する課題に似ていますが、より多くの制約があります。

図 3. さまざまな方法で FPGA アクセラレーター・ボードをデータセンター・ファブリックに接続可能



同様に、仮想化ハイパーバイザーは、FPGA リソースを検出できなければならず、仮想マシンを別の物理 FPGA へ移動することの意味を理解している必要があります。オーケストレーション・ツールと同様に、ハイパーバイザーも均一なファブリックを好みます。データセンター全体で FPGA のワークロード要求、構成、リリースにおいて蓄積される細かな不均一性は、ハイパーバイザーにとって課題となります。

スタックの必要性

多数の重要なアプリケーションにおいて、FPGA アクセラレーションは大幅なパフォーマンス・ゲインをもたらしてきました。しかし、単に FPGA をサーバーラックに追加しただけでは、多くのアプリケーション開発者はこのゲインを利用できません。また、FPGA をデータセンターの MANO 自動化フレームワークに適合させることもできません。

これらの課題に対処するには、適合スタックが必要です。オープン・インターフェイスのガスケットと OS 向けのドライバーも必要です。さらに、さまざまなプログラミングと構成ツール、ライブラリー、開発フレームワーク、ターンキー・ソリューションも必要です。MANO 自動化ツールと主要ベンダーのハイパーバイザーとの緊密な統合も不可欠です。これらすべてを満たして初めて、実現できます。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。