Hot Chips におけるヘテロジニアス時代の前触れ

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「Hot Chips Heralds Heterogeneity」の日本語参考訳です。

シリコンバレーで毎年開催される Hot Chips カンファレンスは、CPU 大手と新興企業の双方のアーキテクチャーに関する考え方について、信頼できる情報を提供しています。2017 年も例外ではなく、アーキテクト達は物理的限界とワークロード要求に取り組み、その年のトレンドであったディープラーニングが特に注目を集めました。多くの論文は、ヘテロジニアス・コンピューティングとハードウェア・アクセラレーターの連携を称賛するものでした (図 1)。

図 1. 高速化は通常さまざまなデバイスの連携により実現される

当然、サーバークラスの CPU に大きな注目が集まりました。AMD、IBM、インテルの各社はそれぞれ最新の製品を発表しました。興味深いことに、AMD Epyc* プロセッサーとインテル® Xeon® スケーラブル・プロセッサーは、前年のカンファレンスで発表されたコア (ぞれぞれ Zen と Skylake†) ベースでした。

表面上は、この 2 つのコアは似ています。どちらも 4 つの整数パイプライン、2 つの浮動小数点パイプライン、アドレス計算ユニット、ロード-ストア・ユニットを備えており、どちらも命令キャッシュからワードをロードして、サイクルごとにいくつかの短い x86 命令を読み取ります。コアは、これらの複雑な可変長の命令をマイクロオペレーションに分割し、データの依存関係を示すためマークして、大きなバッファーへロードします。次に、アウトオブオーダー・ディスパッチ・ユニットは、実行可能なマイクロオペレーションを選択して、利用可能な実行パイプに供給します。

これらはすべて、シングルスレッド・コードから命令レベルの並列性を最大限に引き出すために行われます。新しいプロセス世代に移行するにつれて、トランジスターは多少高速になりますが、その利点はインターコネクトによって相殺され、ブロックレベルの最大周波数は横ばいになります。しかし、新しいノードごとのトランジスター密度は大幅に向上します。そのため、プロセッサー・コアの設計者は、ベンチマークでクロックあたりの有効な命令数を向上できるように、回路上にトランジスターを惜しみなく配置しています。例えば、AMD* の Zen は前世代のコアと比較して、いくつかのコードで IPC が 50% 以上向上したと主張しています。

しかし、単純にパイプラインの数とディスパッチ・ユニットの幅を増やせば良いわけではありません。アーキテクトはストールを回避する必要があります。そうでないと、すべてが無駄になります。それには、これまで以上に複雑な分岐予測、パイプラインのバイパス、予測プリフェッチ、パイプラインの浪費を防ぐ優れたデバイスなど、さらに多くのトランジスターが必要になります。そしてそれは、ほかのすべてが失敗し CPU コアがメモリーを待機するときに失われるクロックサイクル数を最小に抑えるため、より大規模で洗練されたキャッシュ階層を必要とします。

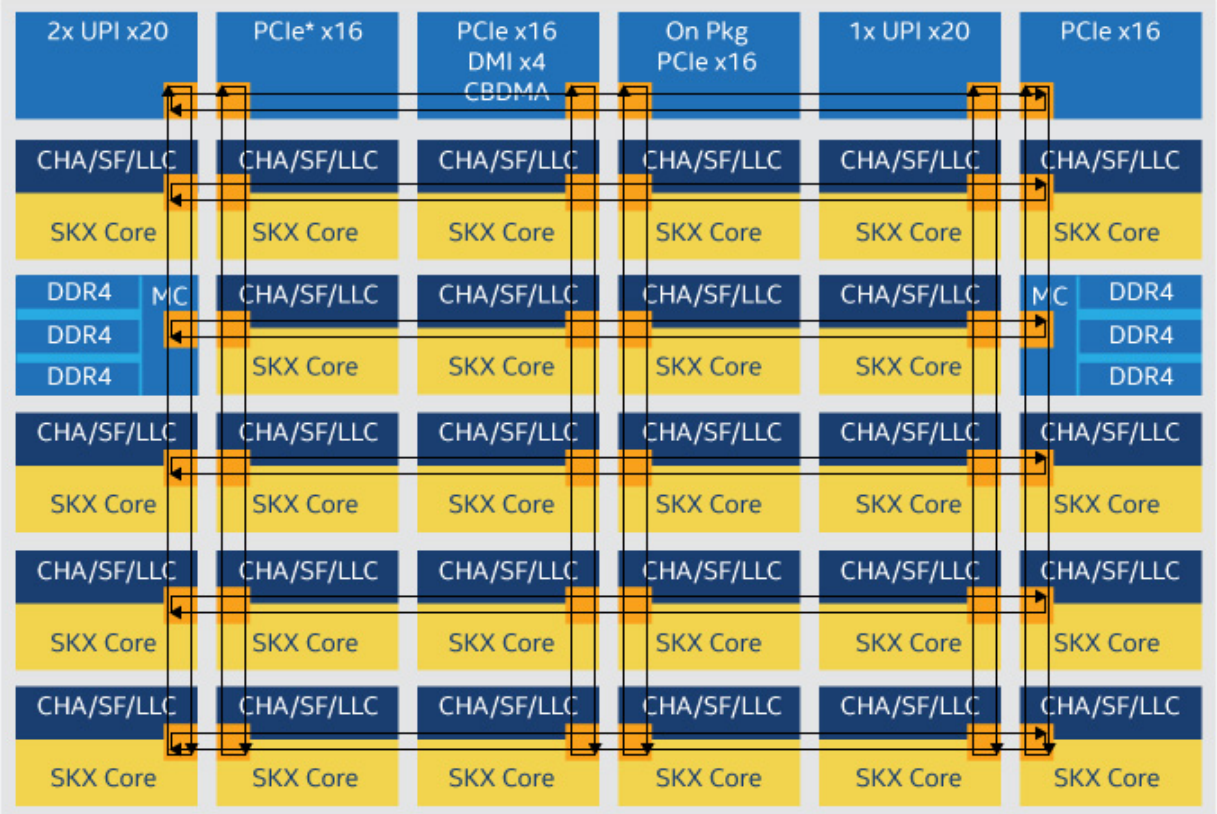
これらはすべて Zen と Skylake† に関する 2016 年の論文で明らかでした。2017 年の論文では、再びより多くのコアをダイに、そしてより多くのダイを回路基板に配置し、多数のコアを同時に実行し続けるのに十分なバス、キャッシュ、およびメモリー帯域幅をシステムに供給しています。

サーバーソケットの内部

AMD Epyc* は、2 つの 4 コア CPU クラスターを搭載した 4 つのダイを有機基盤上に配置します。各コアにはプライベート L2 があり、各クラスターには共有 L3 があります。各ダイには 8 x 72 ビット DRAM チャネルがあり、ダイは独自のポイントツーポイント・ファブリックにより基盤全体にわたって相互接続されています。

一方、インテル® Xeon® スケーラブル・プロセッサーは、1 つのダイに 28 個の Skylake† コアと 6 個の DRAM チャネルを搭載し、メッシュ・インターコネクトを採用しています (図 2)。各コアに L2 があり、L3 は共有です。どちらの設計も、キャッシュのヒット率とキャッシュと DRAM 間の帯域幅を増やすため、多くのダイ領域を必要とします。そして、AMD* の非専用 SerDes 接続やインテル® ウルトラ・パス・インターコネクトのような高速リンクにより、トランジスターはより優れたソケット間の帯域幅を提供しています。

図 2. 多数の Skylake† コアを中心に構築されたインテル® Xeon® スケーラブル・プロセッサー



トランジスターでコアを大きくするのではなく、コア数を増やす主な理由は、リターンを軽減できるからです。あらゆる手を尽くし、強力なコンパイラーによる最適化を利用しても、ほとんどのコードで利用可能な命令レベルの並列性は、3、6、8 発行コアをやや上回る程度であると推定されます。可能な限り速くクロックを実行し、1 クロックごとにソフトウェアで利用可能なできるだけ多くの命令を発行している場合、ほかの並列処理の可能性を見つける必要があります。

クラウド・データセンターでは、並列スレッドにより簡単にパフォーマンスを向上できます。多種多様なワークロードが混在し、その多くはマルチスレッド化されており、大容量のローカルメモリーを備えているため、独立した実行スレッドを多数供給することで、多くのコアをビジー状態にできます。

次に、例えばマップ・レデュース検索やデータを小さなチャンクに分割して各チャンクをスレッドで処理するアルゴリズムを使用して、個々のアプリケーションを多数の独立したスレッドで実行したらどうでしょうか? 確かに、メモリー帯域幅を使い果たさなければ、多くのコアを使用することでスレッドを多用するアプリケーションのスピードアップが可能です。しかし多くの場合、多数のスレッドを独立したプロセッサー・コアに割り当てるだけでは不十分です。

マルチコアの有用性

前述のとおり、サーバーチップのアーキテクトは各ソケットがそれらすべてのコアをサポートするのに十分な DRAM 帯域幅と容量にアクセスできるようにするため、多大な努力を払ってきました。しかし、1 ソケットの利用可能な帯域幅で、28 コアや 32 コア以上を使用する可能性がある、スレッドを多用する計算負荷の高いワークロードに注目すると、興味深いトレードオフに直面します。個々のコアを単純化 (縮小) すると、各コアのシングルスレッド・パフォーマンスは低下しますが、ソケットごとのコア数が大幅に増加する可能性があります。計算負荷の高いコアでは、内部ループが縮小された L1 命令キャッシュに収まる場合、大きな利点をもたらします。これが、インテルの Knights Mill† メニーコアチップと多くの急進的なプレゼンテーションの背後にある理論です。

Knights Mill† は、インテル® Xeon® プロセッサー互換コアを 2 つの発行エンジン (アウトオブオーダー実行) に単純化し、マシンラーニング向けのいくつかの機能を追加しています。オンダイ L3 キャッシュを排除して、代わりに 16GB 高速 DRAM をデバイスのマルチダイモジュールに搭載しています。この変更により、アーキテクトは 72 コア、L1 および L2 キャッシュ、ベクトル・プロセシング・ユニット (VPU) をすべてメインダイに配置できます。

Knights Mill† は、サーバー CPU とハードウェア・アクセラレーターの中間に位置します。インテル® Xeon® プロセッサーで動作する標準のソフトウェアとワークロードを実行できますが、特にメニーコア・プロセッサー向けの数値演算負荷の高いワークロードを高速化するように最適化されています。そのため、純粋なアクセラレーターとは異なり、一部のコードの書き直しが必要で、CPU と連携して使用する必要があります。

Baidu* (百度) の論文は、このアイデアをさらに特殊化したものです。合理化された x86 コアの代わりに、Baidu* は小さなコアを作成し、大規模な FPGA に数千近くのコアを収められるようにしました。このサイズでは、コアを汎用にすることは困難なため、代わりに Baidu* はそれらに小さなドメイン固有の命令セットを提供しました。これは、FPGA にとっては魅力的な代替手段ですが、ASIC にとってはそれほど実用的ではありません。課題はまだたくさんあると認めながらも、論文では FPGA に 256 コアを実装することで、8 コアのインテル® Xeon® プロセッサー 1 基と比較してタスクが最大 64 倍高速化されたとしています。

このテーマについては、コーネル大学、カリフォルニア大学ロサンゼルス校 (UCLA)、カリフォルニア大学サンディエゴ校、ミシガン大学のチップ設計者で構成された別のチームでも研究が行われました。このチームは、5 個のハイエンド RISC-V* Rocket コアから成る汎用層と 496 個の小さな RISC-V* Vanilla-5 コアから成るアクセラレーター・アレイの 2 種類のオープンソースの RISC-V* コアに加えて、専用のニューラル・ネットワーク・アクセラレーション・ハードウェア層を持つプロセッサー階層を設計しました。

データ並列処理の利用

単一命令シーケンスの多数の同一コピーにデータを分割して多数のスレッドを使用する場合、冗長な回路を排除し、さらに多くの実行ユニットを配置するためダイ面積と消費電力を抑える方法があります。例えば、すべてのコアが同じ命令シーケンスを実行し、それらが必ずしもロックステップではない場合、単一の命令キャッシュ、デコードユニット、リオーダーバッファーを使用し、実行ユニットごとに個別のディスパッチ・バッファーと命令カウンターを保持することができます。これは、シングルスレッド、複数データ (STMD) と呼ばれます。このアプローチは、高度なグラフィックス・プロセシング・ユニット (GPU) に近く、そのうちの 2 つは Hot Chip の論文で紹介されています。

各実行ユニットが同一の命令シーケンスを実行する場合、単一のフェッチ、デコード、ディスパッチ・パイプラインをすべての実行ユニットで共有できます。これは、単一命令、複数データ (SIMD) と呼ばれます。最もよく知られている SIMD 実装は、従来の GPU シェーディング・エンジン・アレイであり、これは今日の高度な GPU の内部動作でも利用されています。専用のグラフィックス・プロセシング・ハードウェアとともに、GPU には多数の小さな浮動小数点プロセッサーが含まれており、通常それらは SIMD クラスターのグループとしてまとめられています。これらの小さなプロセッサーは、元々 3D イメージ・レンダリングにおける多角形のシェーディング・アルゴリズムを実行するためだけに設計されましたが、ほかの計算負荷の高いコードを扱えるように汎用化されました。膨大な数の小さな浮動小数点エンジンと非常に大きなメモリー帯域幅により、GPU はハイパフォーマンス・コンピューティングと計算負荷の高いワークロードを処理するデータセンターで最も広く使用されるアクセラレーターとなりました。後者は、計算負荷が高く、データ並列タスクを使用するマシンラーニングへの関心の高まりとともに、急速に拡大しています。

Hot Chips で紹介された AMD* Radeon* Vega 10 に関する論文と NVIDIA* Tesla* V100 に関する論文は、GPU がマシンラーニングのワークロードの特性に積極的に適応しようとしていることを示しています。主な変更の 1 つはデータパスの幅です。近年の研究により、訓練後にネットワークを使用して入力を分類するディープラーニングの推論では、GPU 実行ユニットの完全な 32 ビット浮動小数点精度を必要としないことが分かりました。AMD* と NVIDIA* はそれぞれ小さなエンジンにパックド 16 ビット・データ型を追加しています。また、データを速く取り込むという 3D レンダリングとディープラーニング・ネットワーク計算の両方の要件に対応するため、どちらもインパッケージの高帯域幅メモリー (HBM) を採用しています。

さらに NVIDIA* は、膨大な数の小さなコアがディープラーニング計算の中心である行列演算にとって理想的ではないことを認識し、おそらく Google* の Tensor Processing Unit* (TPU) ASIC を考慮して、行列積和演算を高速化するため、V100 に 640 個のテンソルコアを追加しています。各テンソルコアは、16 ビット浮動小数点オペランドを使用して 32 ビット浮動小数点合計を生成する、4 x 4 行列の固定積和演算データパスを提供します。テンソルコアは、例えば 16 x 16 行列の積和演算を実行するため、ワープにまとめることができます。その結果、NVIDIA* は 16 ビット浮動小数点行列演算において、前世代の P100 GPU と比較して 9 倍のスピードアップを達成しました。

GPU にアプリケーション固有のロジックを含めることで、NVIDIA* は Google* の Tensor Processing Unit* (TPU) という、全く異なる種類のアクセラレーターからヒントを得ました。TPU も 2017 年の論文で関心の高かった主題です。TPU は、基本的に精巧な CPU インターフェイスを備えたハードウェア行列乗算器/アキュムレーターです。本格的な計算は、8 ビット乗算器の 256 x 256 のシストリック・アレイで処理されます。

行列演算以外に考慮すべきこと

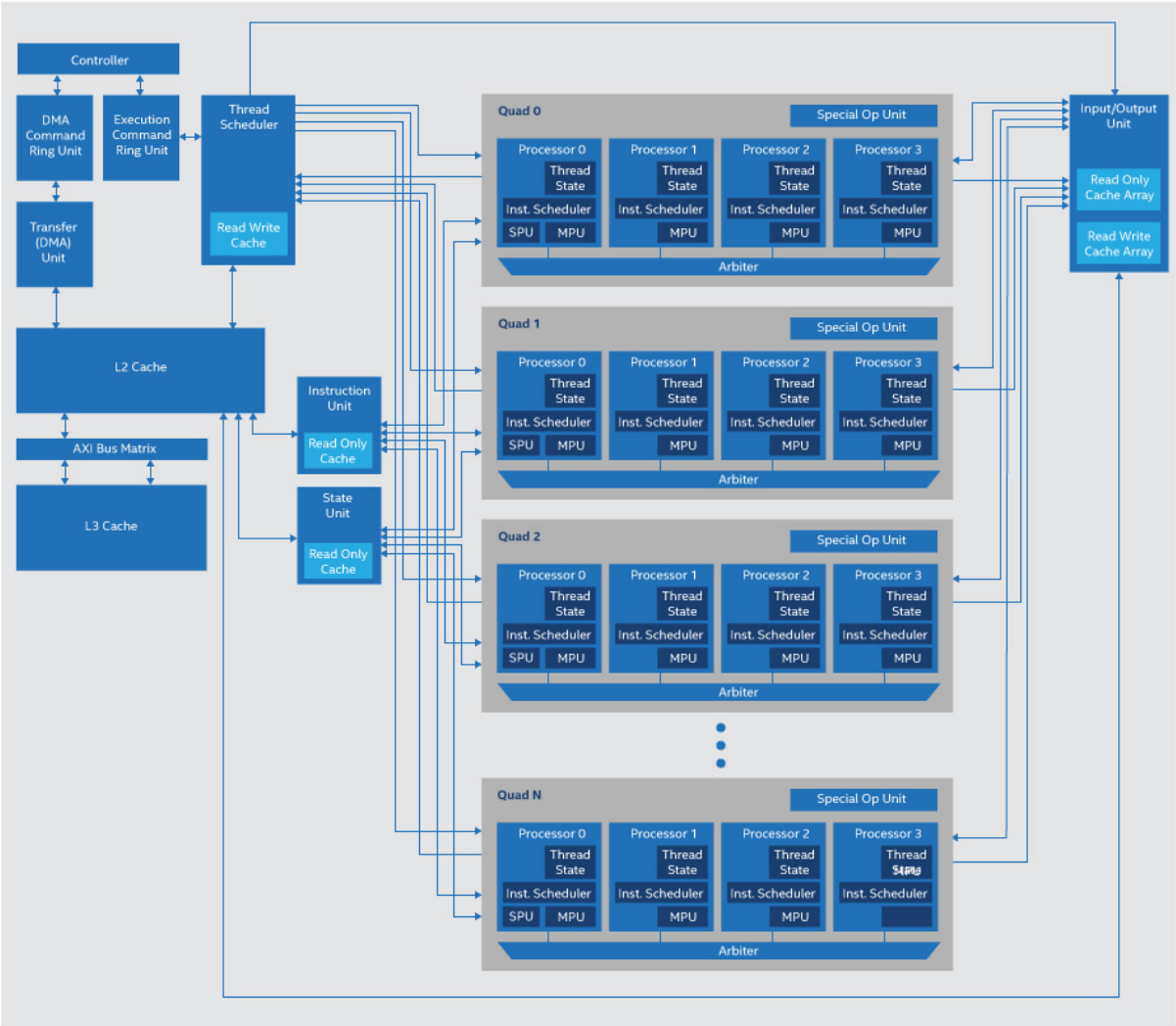
ディープラーニングの中核は人工ニューロンであり、本質的にはベクトルのドット積であると認識すると、ニューラル・ネットワーク・アクセラレーションは、低電力の行列乗算器によって高速化される線形代数と考えられます。しかし、ARM* Research、ハーバード大学、プリンストン大学の共同論文は、この考え方は実際に推論ネットワークで行われていることを正しく理解していないと主張しています。ターゲット・アプリケーションがディープラーニングの推論であり、スペースと電力の制約があるネットワーク・エッジで実行される場合、すべての行列のスパース性を利用する必要があると論文は主張しています。ディープラーニング・ネットワークが訓練を完了する

と、多くの重み (活性度と呼ばれる、各ニューロンで入力に乗算される係数) はゼロまたはほぼゼロになります。そのため、ニューロンの出力となるドット積を計算すると、個々の積の多くは問題にならないほど小さくなります。ARM* の論文は、これらの不必要な操作を排除するアプローチについて述べています。その設計では、静的スケジューラーは重要な重みをローカルメモリーにキャッシュして、チップの多数の実行ユニット・パイプラインを通して必要な操作をルーティングし、不要な操作を排除します。

しかし、プルーニングされるネットワークを実行するまで情報がない訓練についてはどうでしょうか? 多くの命令とロード/ストア操作を排除可能な 1 つの手法は、コードのデータフロー・グラフをハードウェアに直接マップすることです。このようなデータフロー・グラフは、多くの場合、コンパイラーで中間形式として使用されており、多くのマシンラーニング・フレームワークの基盤であるため、簡単に利用できます。そして、最適化とプルーニングにはオープンソースのユーティリティーを利用できます。

Hot Chips では、2 つの論文でデータフロー・グラフをシリコンに直接マップする全く異なるアプローチが紹介されました。1 つは Thinci* (「think-eye (シンクアイ)」と発音) によるもので、サーバー CPU が命令スケジューラーによって管理される実行ユニットのプールであるように、Thinci* のチップは基本的にスレッド・スケジューラーによって管理される小さなプロセッサーのプールです (図 3)。このチップは、データフロー・グラフをスレッドのセットに分解し、各スレッドはグラフ中の 1 つのノードの計算を表します。そして、プロセッサーが中間 RAM へのロードとストアを繰り返すのではなく、互いに直接データをストリーミングできる方法で、これらのスレッドをプロセッサーにマップします。

図 3. Thinci* のグラフ・プロセッサーはスレッド (データフロー・グラフ・ノード) をプロセシング要素のアレイへマップ



より急進的なアプローチが Wave Computing* により紹介されました。Wave Computing* の論文は、プログラム可能なスイッチマトリクスで相互接続された最大 16K プロセシング要素のアレイに関するものでした。コンパイラーは、データフロー・グラフをこのアレイに直接マップして、グラフノードをプロセシング要素に割り当て、スイッチを介してノードを相互接続します。そのため、ある意味、グラフはチップのネイティブ言語と言えます。Wave Computing* では、自動化ツールと IC により、手動でグラフを高度な FPGA へマップした場合と同等のアクセラレーションとダイ面積が得られるとしています。

アクセラレーターの台頭

計算負荷の高いカーネルで命令フェッチと中間データストレージを軽減または排除し、データ並列性とスレッド並列性を利用し、膨大な量の演算器をダイに配置できるため、ハードウェア・アクセラレーター (メニーコア CPU、GPU、演算アレイ、データフロー・エンジン、または FPGA) は、コンピューティングにおいて恒久的な役割を得つつあります。実際、2017 年に開催された Linley Processor Conference においてアナリストの Linley Gwennap 氏は、「今後、アクセラレーターはワークロードの大部分を処理するようになるでしょう。」と述べています。

だからと言って CPU が何もしなくなるわけではありません。プロセス・テクノロジーによって数年ごとに新しいトランジスターが生み出されるのですから。CPU アーキテクトは、ベンチマークと重要な顧客コードに残された命令

レベルとスレッドレベルの並列性を利用する新たな方法を見つけるでしょう。しかし、彼らがアクセラレーターの能力を無視することはないでしょう。

ますます洗練されたベクトル・プロセシング・ユニット (サーバー CPU チップで以前から利用可能な浮動小数点 SIMD エンジン) に加えて、一部のサーバー CPU は暗号化エンジンも搭載しています。また、より汎用的なプロセッサーは、汎用機能とディープラーニング機能を備えた統合 GPU を搭載しています。そして一部のエキスパートは、今後さらに多くの種類の統合ハードウェア・アクセラレーションが登場すると予想しています。1 つ確かなことは、モバイルデバイスからクラウドに至るまで、アプリケーション開発者はさまざまなヘテロジニアス・プラットフォームを利用できるようになるでしょう。

† 開発コード名