SYCL* でヘテロジニアス・プログラミングの課題を解決する

この記事は、HPCwire に公開されている「Solving Heterogeneous Programming Challenges with SYCL」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

C++ のヘテロジニアスな将来の考察

本記事の著者である James Reinders は、完全なヘテロジニアス・コンピューティングへの進化は、オープンで、マルチベンダーかつマルチアーキテクチャーのアプローチにおいて最も利点をもたらすと考えています。そして、インテルがこのオープンな将来の実現に大きく貢献できると信じ、1 年前にインテル コーポレーションに戻ってきました。この記事では、SYCL* をさまざまな角度から掘り下げ、さらに理解を深めるための提案やヒントを提供します。

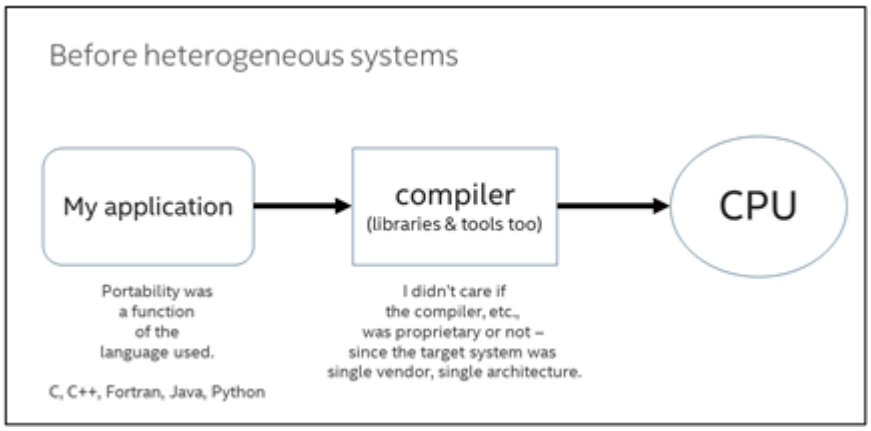
SYCL* は、C++ に完全なヘテロジニアス・データ並列処理をもたらす Khronos の標準規格です。その重要性を説くことは簡単ですが、理解することは容易ではありません。SYCL* は万能薬ではありませんが、今後爆発的に多様化するハードウェアに対し、どのようにプログラミングするかという、重大な問題に対する解決策なのです。

カンブリア爆発に直面したプログラミング

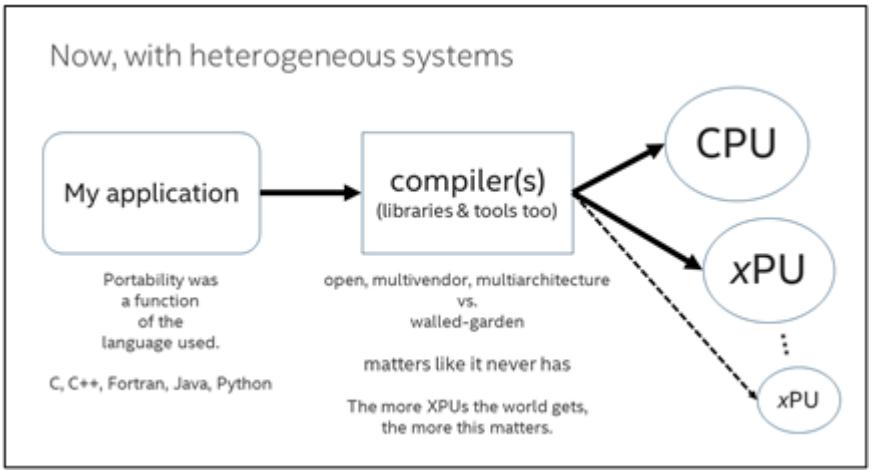
John Hennessy 氏と David Patterson 氏は、「コンピューター・アーキテクチャーの新たな黄金期」(英語) を迎えようとしている理由を説明し、「次の 10 年は、新しいコンピューター・アーキテクチャーのカンブリア爆発が起こり、学術界と産業界のコンピューター・アーキテクトにとってエキサイティングな時代になるだろう」と期待しています。

アプリケーションにパフォーマンスをもたらすため、CPU、GPU、FPGA、AI チップ、ASIC、DSP、その他のイノベーションが注目を集めることになるでしょう。また、これまでにない方法で複数のダイを 1 つのパッケージに統合する技術革新も高まっており、さらなる多様化による計算処理の高速化と相まって、途方もないチャンスであると言えます。

プログラミングには、すべてのハードウェアをサポートするシンプルなソリューションが求められます。これは、NVIDIA* CUDA* のような、独自の標準や技術の世界では決してもたらされません。



何十年もの間、プログラマーはシステム内の単一種類のデバイスのパフォーマンスを引き出すコードを記述し、ツールはその単一種類のデバイスで最高のパフォーマンスを提供できれば良かったのです。



ハードウェアの多様化が進むと、1 つのアプリケーションで複数の種類のデバイス (XPU、つまり、あらゆるベンダーのあらゆるアーキテクチャーの計算ユニット) を計算に使用することが期待されます。そのため、これまでの単一種類のデバイスに特化したツールチェーンや言語では対応できません。

将来的には、完全にヘテロジニアスなシステムのあらゆる能力を最大限に引き出すことができないツールや言語は、淘汰されることになるでしょう。オープンで、マルチベンダー、かつマルチアーキテクチャーをサポートすることが求められます。つまり、ツール、ライブラリー、コンパイラー、フレームワーク、およびソフトウェア開発者が必要とするものが、標準で XPU (あらゆるベンダーの、あらゆるアーキテクチャーの、あらゆる計算ユニット) をサポートする必要があります。

カンブリア爆発に SYCL* を役立てる方法

ヘテロジニアス・マシンのプログラミングには、2 つのものが必要です。1 つは、アプリケーションで利用可能なすべてのデバイスを実行時に検出する方法で、もう 1 つは、アプリケーションの処理を支援するためデバイスを利用する方法です。

SYCL* は、最新の C++ をベースに構築されており、SYCL* の 2 つの基本機能である queue と queue.submit によって、ヘテロジニアス・プログラミングの 2 つの課題を解決します。

SYCL* の queue

SYCL* の queue を構築すると、1 つのデバイスへの接続が作成されます。デバイス選択のオプションは、次のいずれかです。

a. ランタイムが選ぶデフォルトを受け入れる
b. 特定のクラスのデバイス (GPU、CPU、FPGA など) を要求する
c. 追加のヒントを指定する (例えば、ユニバーサル共有メモリーや FP16 のデバイス割り当てをサポートするデバイス)
d. 関数を記述して完全に制御し、利用可能なすべてのデバイスを調べ、スコアを付ける。

```
// デフォルトのセレクターを使用する
queue q1{}; // default_selector
queue q2{default_selector()};

// CPU を使用する
queue q3{cpu_selector()};

// GPU を使用する
queue q4{gpu_selector()};

// 追加のヒントを指定する
queue q6{aspect_selector(std::vector{aspect::fp16,
aspect::usm_device_allocations})};

// 関数を記述して、より複雑な選択を使用する
queue q5{my_custom_selector(a, b, c)};
```

デバイスに接続する SYCL* キューの構築は、任意の精度レベルで行うことができます。

SYCL* の queue.submit

queue を作成したら、ワーク (「カーネル (コード)」と呼ぶ) を投入できます。ワークの実行順序は、既知の依存関係 (例えば、データは消費される前に作成される必要がある) に違反しない限り、ランタイムに任されます。プログラミング・スタイルに応じて、インオーダーキューを要求するオプションもあります。

```
// queue.submit コード
q.submit([&](handler &h) {
    h.parallel_for(num_items,
        [=](auto i) { sum[i] = a[i] + b[i]; }
    );
});


// ショートバージョン
q.parallel_for(num_items,
    [=](auto i) { sum[i] = a[i] + b[i]; }
);
```

ワークがキューに投入され、キューがデバイスに接続されます。集計を行うコード行はデバイスで実行されます。

並列プログラミングの効率良いアプローチであるカーネル・プログラミング

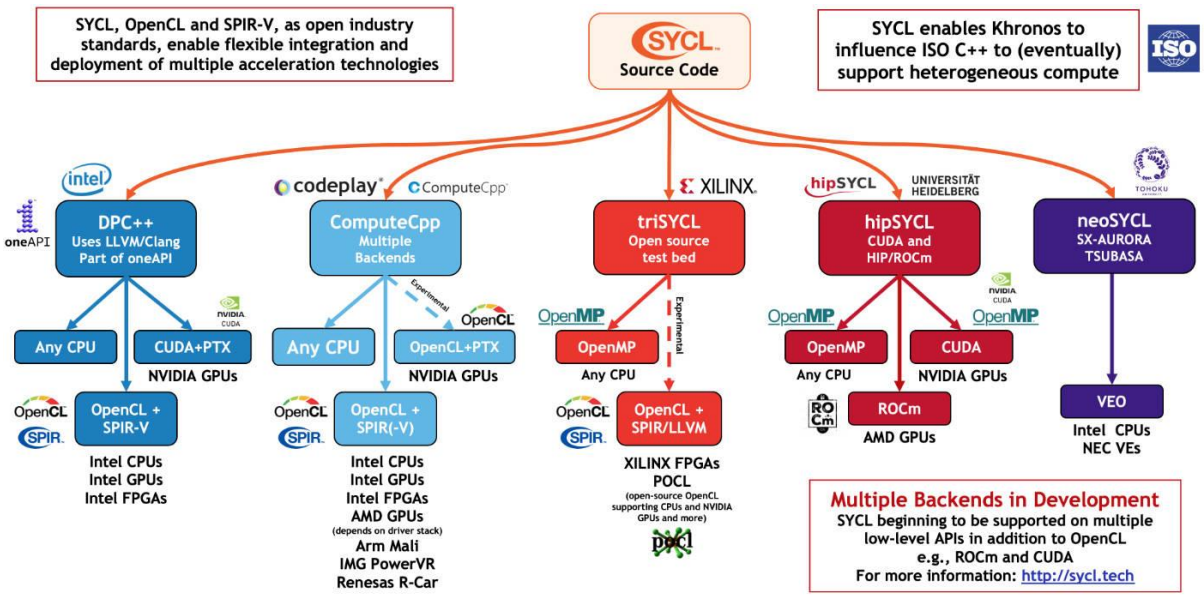
カーネル・プログラミングは、上記の例の集計のような単純な操作を記述し、デバイスに「該当するすべてのデータに対してその操作を並列に実行する」ように効率良く伝えることができるため、並列処理を表現するには

最適な方法です。カーネル・プログラミングは、シェーダー・コンパイラー、CUDA*、OpenCL* に見られる確立された概念です。最新の C++ は、上記のコード例に示すように、ラムダ関数でこれをサポートしています。

SYCL* 実装

LLVM に SYCL* を実装するプロジェクトは、データ並列 C++ (DPC++) と名付けられました。LLVM は、素晴らしいコンパイラー・フレームワークです。AMD、Apple、IBM、インテル、NVIDIA など多くの企業が、LLVM を使用したコンパイラーに移行 (英語) しています。

DPC++ だけが利用可能な実装ではありません。SYCL* は広くサポートされており、現時点では DPC++、ComputeCpp*、triSYCL、hipSYCL、neoSYCL の少なくとも 5 つのコンパイラーが SYCL* サポートを実装しています。SYCL* を実装する初の LLVM プロジェクトである DPC++ はインテル、ComputeCpp* は Codeplay、triSYCL は AMD と Xilinx、hipSYCL はハイデルベルク大学、neoSYCL は NEC によって開発されました。



SYCL* の実装数の増加

図の出典: https://www.khronos.org/sycl (英語)

GPU 以外のアクセラレーターの使用

パフォーマンスを最大限に引き出すため、アプリケーションではすでに特定のデバイス向けに主要ルーチンを特化させる作業を行っています。例えば、ライブラリーは、CPU や GPU ごとに異なる実装を選択することがよくあります。

SYCL* は、クロスアーキテクチャーのコードを記述する方法を提供する一方で、正当化されると判断した場合には特殊化を可能にします。

デバイスを列挙する SYCL* サポートには、プラットフォームとバックエンド、およびそれらの処理能力を調査する機能が含まれています。オープンで、マルチベンダー、かつマルチアーキテクチャーであることが重要です。

多くの SYCL* アプリケーションは、デバイスセレクターとデバイスクエリーを使用してパラメーターを調整し、多くのデバイスで実行できる汎用的なプログラム構造とカーネルコードを採用しています。プログラミングの抽象化により、移植性に優れたプログラムを記述できますが、デバイスの処理能力によっては、デバイスを最大限に活用するため、アプリケーションの一部を書き換える必要があります。

コードの移植性とパフォーマンスの移植性は重要であり、ヘテロジニアスの世界における多種多様な要求は大いに注目されています。SC21 に合わせて開催されたベスト・プラクティスを共有するコミュニティー P3HPC (英語) — Performance, Portability & Productivity in HPC (HPC におけるパフォーマンス、移植性、生産性) — のワークショップでは、さまざまなトピックについてプレゼンテーションが行われました。興味のある方は、最先端のコードの移植性とパフォーマンスの移植性の測定に関する「Navigating Performance, Portability, and Productivity」(英語) を参照してください。

C++ 並列処理

SYCL* は、テンプレートやラムダなどの標準 C++ の機能をベースにした最新の C++ であり、新しいキーワードや言語機能を学ぶ必要はありません。SYCL* に対応するように C++ コンパイラーを拡張することで、パフォーマンスを向上させる最適化が可能になり、複数のバックエンドを自動的に呼び出して、1 回のビルドで任意の複数のアーキテクチャー向けの実行ファイルを作成できます。

SYCL* は、ヘテロジニアス・プログラミングの重要な課題に解決策を提供します。例えば、さまざまなコヒーレンシー・モデルを持つローカルメモリーとリモートメモリーを管理する方法や、多様な計算能力を検出する方法、特定のデバイスに作業を割り当てて必要なデータを供給し、その結果を管理する方法などがあります。

C++23 は、並列プログラミングをサポートする C++ の進化版です。p2300 (英語) の説明では、「std::execution」の目的は、構造化された非同期実行の基礎的なサポートを提供することです。当然のことながら、C++23 はヘテロジニアス・プログラミングのすべての課題を解決しようとするものではなく、解決しようとする必要もありませんし、そうしないことで批判されることもないでしょう。C++23 は CUDA* や SYCL* の代替ではなく、CUDA* や SYCL* の必要性は変わらないでしょう。

SYCL* は、多くのベンダーの、さまざまなアーキテクチャーのハードウェアを有効利用できる方法で、これらの問題を解決します。この姿勢は、C++ だけでなく、Python* を含むほかの標準化の取り組みにも反映されるでしょう。SYCL* は、プログラマーが直面している課題を解決し、ともに発展していくでしょう。

オープン、マルチベンダー、マルチアーキテクチャー

ハードウェアの多様性の力を信じ、差し迫ったカンブリア爆発を利用したいと考えているなら、SYCL* を試してみる価値があります。オープンで、マルチベンダー、かつマルチアーキテクチャーのアプローチはほかにもありますが、C++ プログラマーにとって SYCL* は重要です。

SYCL* は魔法ではありませんが、C++ ユーザーがコンピューター・アーキテクチャーの新たな黄金期に対応するための確かな一歩と言えます。プログラマーとして、アプリケーションの柔軟性を保つことで、ハードウェアの多様性を促進させることができます。SYCL* は、ニーズに合わせてコードの多くを共通に保ちながら、それを実現する方法を提供します。

関連情報

実際に試してみることで多くのことを学べます。https://sycl.tech/ (英語) には、オンライン・チュートリアルや SYCL* 書籍へのリンク (無料でダウンロード可能な PDF (英語) もあります)、現在の SYCL* 2020 標準仕様へのリンクを含む、SYCL* の習得に役立つ情報があります。XPU ブログ (英語) では、インテル® DevCloud (インテルの各種 CPU、GPU、および FPGA を利用できる無料のオンラインアカウント) へのアクセス方法と複数の SYCL* コンパイラー (英語) に関する役立つヒントを紹介しています。これらの手順に従えば、数分で最初の SYCL* プログラムをコンパイルして実行できます。

著者紹介

本記事の著者である James Reinders は、完全なヘテロジニアス・コンピューティングへの進化は、オープンで、マルチベンダーかつマルチアーキテクチャーのアプローチにおいて最も大きな利点をもたらすと考えています。そして、インテルがこのオープンな将来の実現に大きく貢献できると信じ、1 年前にインテル コーポレーションに戻ってきました。並列プログラミングに関する 10 冊の技術書の著者 (または共著者、編集者) であり、最新の著書は SYCL* に関するものです (ここから (英語) 無料でダウンロードできます)。