SYCL* を使用すべき理由

この記事は、HPCwire に公開されている「Why SYCL: Elephants in the SYCL Room」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。

解説 — ヘテロジニアス・コンピューティングに関する寄稿記事の第 2 弾として、短期間の「リタイア」を経て 2021 年にインテルに復職した James Reinders が、前回の記事「SYCL* でヘテロジニアス・プログラミングの課題を解決する」に続いて、SYCL* が C++ のヘテロジニアスな将来にどのように貢献するかを述べます。今回は、SYCL* 委員会の現委員長でもある Codeplay Software Ltd. の Michael Wong 氏と一緒に、「SYCL* に関する重要な疑問」について考察します。

C++ プログラミングと SYCL* による完全なヘテロジニアス・サポートについては、前回の記事「C++ のヘテロジニアスな将来の考察」をはじめ、SYCL* 仕様の関係者や支持者が説明しており、sycl.tech (英語) にも数多くのリソースが掲載されています。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。SYCL* は万能薬ではありませんが、「爆発的に多様化するハードウェアに対し、ヘテロジニアス・プログラミングを適切に実現する」という重大な課題に対応します。

この記事では、SYCL* に関する重要な疑問について、この分野に何十年も携わってきた私たちの見解を皆さんと共有します。これらの疑問は、SYCL* が自分にとって重要かどうかを、ソフトウェア開発者が理解するためのものです。これまで、重要なプロジェクトはすべて、ある時点でこのような疑問 (Elephants in the Room[1]) に向き合い、オープンに対応しています。

重要な疑問 1: GPU で十分ではないのか? ほかのアクセラレーターは必要なのか?

どのアクセラレーターが生き残り、どのアクセラレーターが一時の流行になるかは当然考えるべきことです。何十年もの間、さまざまなアクセラレーターが登場しては消える中、CPU は存続してきました。今日、GPU はコンピューター・システムの大部分に搭載されています。GPU を活用するアプリケーションを記述することは、GPU がほぼどこにでもあることを考えると、非常に理にかなっています。

最初の重要な疑問は、本当に一般化する必要があるのか、つまり、マルチアーキテクチャーかつマルチベンダーにする必要があるのか、ということです。

近藤正明教授を中心とした研究者による「次世代先端的計算基盤に関する白書」や John Hennessy 氏と David Patterson 氏による「A New Golden Age for Computer Architecture (コンピューター・アーキテクチャーの新たな黄金期)」 (英語) で述べているように、専門家は今後 10 年間に「専用または半専用のハードウェア・アクセラレーター」がコンピューティングの必須機能になると予想しています。

専用アクセラレーターは、GPU に限定されません。多種多様なアクセラレーター向けに最適化することが目的ですが、アクセラレーターごとに異なるコードを記述したいわけではありません。誰もが貢献し、協力し、特定のベンダーに縛られることなく、メンバーや一般の要求に基づいて有機的に進化できる標準化された言語が、業界に利益をもたらします。

SYCL* では、必要に応じて共通コードと特殊コードを使い分けることができます。そうすることで SYCL* は幅広いアクセラレーターに対応し、必要に応じてプログラマーの判断で共通のクロスアーキテクチャー・コードとアーキテクチャー固有の特殊コードを記述できます。

その基礎となるプログラミング・モデルである SPMD は、多くのアーキテクチャーで有用であることが示されています。SPMD は、NVIDIA* CUDA*/OpenCL*/SYCL* を使用するほとんどのプログラマーが考える、1 つのワークアイテムを操作するコードを記述して、複数のワークアイテムがベクトル・ハードウェア・レーンを埋めるように、ほとんどのハードウェア上で同時に実行されることが期待されます。

SYCL* は、ベンダー (例えば、多くの異なる GPU) やアーキテクチャー (例えば、GPU、FPGA、ASIC) 間で高い移植性を提供します。

重要な疑問 2: NVIDIA* CUDA* を使用すれば良いのでは?

複数の GPU ベンダーから新たな GPU エコシステムが登場し、アクセラレーター市場の競争はますます激しくなっています。NVIDIA* GPU を利用する CUDA* アプリケーションのインストール・ベースは、NVIDIA* だけでなく、すべてのベンダーに対応するために作成された、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ソフトウェア・アプローチに時間をかけて適応できます。

CUDA* は、エコシステムにおける価値提案と NVIDIA* GPU の強さから支持されていますが、CUDA* の囲い込みに対する懸念も高まっています。このような懸念は、以下のような CUDA* の独自性から生じています。

1. CUDA* の定義、実装、進化は、NVIDIA* GPU 製品の設計に役立つように NVIDIA によって管理されています。CUDA* の新機能の詳細は、通常、NVIDIA のハードウェアとソフトウェアの両方がそれをサポートするまで一般に公開されません。以下で詳しく説明するように、この管理体制は、ほかのベンダーのイノベーションを阻害します。
2. NVIDIA の CUDA* ツールおよびライブラリーのライセンス (英語) には、「NVIDIA* GPU を搭載したシステムで使用するアプリケーションのみを開発する」ために使用しなければならないことが明記されています。NVIDIA の「オープンソース」でも、同様に主要部分を制限するライセンス条項 (英語) が含まれています。

NVIDIA* CUDA* は、NVIDIA* GPU による高速コンピューティングを大衆にもたらしました。

アクセラレーター市場における競合製品の爆発的な増加により、ますますオープンで透明性が高まる世界において、CUDA* は壁で囲まれた庭のように見えるかもしれません。

CUDA* に代わるオープンで、マルチベンダーかつマルチアーキテクチャーのソリューションへの要望がなくなることはないでしょう。

重要な疑問 3: AMD* HIP を使用すれば良いのでは?

AMD* HIP (Heterogeneous-Computing Interface for Portability) は C++ 言語方言です。AMD のツールには、CUDA* コードから HIP への移行を支援する「HIPify ツール」があります。AMD は、「HIP コードは、(HCC コンパイラーを利用して) AMD* ハードウェア上または (NVCC コンパイラーを利用して) NVIDIA* ハードウェア上で実行でき、元の CUDA* コードと比較してパフォーマンスの低下はありません」(英語) と述べています。

HIP は「CUDA* 追従」戦略を取っており、AMD は NVIDIA* が CUDA* プラットフォームのアップデートをリリースした後に HIP のアップデートを開発してリリースしています。HIP の利点は、大規模な CUDA* コードベースを AMD* GPU 向けに再利用できることです。しかし、CUDA* の不透明性により、CUDA* を詳しく、タイムリーかつ正確に追うことは困難です。このため、CUDA* 開発者が AMD 独自のハードウェア・イノベーションを利用するには、#ifdef で AMD* GPU 用にコードを切り替える必要があります。

AMD は、NVIDIA* GPU の代わりに AMD* GPU を利用できる HIP を提供しましたが、開発者がそれ以上を望むのは無理もないことです。CUDA* の機能革新とパフォーマンスに遅れをとらず、ついていくことができるソリューションを想像してみてください。

壁に囲まれた庭ではなく、自然あふれた野原でこそ、イノベーションは最も花開くと信じています。

[編集部注: HIP 上で ROCm を実行している AMD* GPU と NVIDIA* GPU をターゲットにした hipSYCL (英語) という SYCL* 実装もあります。]

重要な疑問 4: OpenCL* を使用すれば良いのでは?

OpenCL* は、オープンなマルチベンダーの代替手段を提供しますが、SYCL* や CUDA* よりもソフトウェア・スタックの低層で提供されます。SYCL* は、ヘテロジニアス並列アーキテクチャー向けの標準 C++ インターフェイスを提供することで、OpenCL* のオープン、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の利点を実現することを目的として生まれました。SYCL* の実装には OpenCL* を利用することが多く、SYCL* 2020 ではほかのバックエンドも柔軟に利用できます。SYCL* は、C++ の抽象化を利用して、OpenCL* の利点をより生産性の高い形で実現します。

重要な疑問 5: C++ を使用すれば良いのでは?

ヘテロジニアス・マシンをプログラムする必要があり、移植性は重要だが、移植性のためにパフォーマンスを犠牲にしたくない、と仮定します。

この場合、SYCL* のサポートがあれば、**C++ で十分**だと言えるかもしれません。C++ は、SYCL* のようなテンプレート・ライブラリーによって拡張されることを前提に作られています。SYCL* は新しいキーワードを追加しませんが、SYCL* 対応の C++ コンパイラーは、クロスコンパイル、ファットバイナリー、リモートメモリーをサポートします。これらは、C++ コンパイラーでは歴史的に容易でなかったものです。

SYCL* はまた、ISO C++ で構築されたヘテロジニアス・コンピューティング向けのプログラミングに対応するソリューションを、標準 C++ の中で提供しています。これには、デバイスの列挙 (info)、ワークの定義 (kernels)、デバイス間のワークの送信と調整 (queue)、リモートメモリーの管理などが含まれます。

これが、**C++ では十分ではない**理由です。C++ 標準規格では、連続しない (コヒーレントでない) メモリーを持つヘテロジニアス・システムのサポートは定義されていません。今後追加されるだろうという意見があり、その方向で取り組みも行われていますが、関係者でさえ対応には少なくとも 10 年 はかかると考えており、C++ は何百万行もの既存のコードとの後方互換性を維持する必要性があることがネックになっています。SYCL* 委員会のメンバー (Michael Wong) が、C++ の取り組みを促す論文を執筆しましたが、WG21 (ISO C++) (英語) の対応は、後方互換性の懸念から、メモリーとアドレス指定モデルを根本的に変えるのではなく、並列アルゴリズムとエグゼキューターから始めて、進展させるというものでした。したがって、ヘテロジニアス・マシン向けにプログラミングする場合、「C++ で十分だ」と主張するのは無理があります。さまざまな方向に進もうとする動き

があり、それが競争によってもたらされる利点であり、何が市場や消費者のためになるかを見極められます。今すぐ利用できるのは、「C++ + SYCL*」、「C++ + CUDA*」、「C++ + OpenCL*」でしょう。

C++ コンパイラーとランタイムに SYCL* サポートを追加する目的は、現在提供されていない C++ による完全なヘテロジニアス・サポートを実現する機能を追加することです。また、ISO 標準は既存の知識のベスト・プラクティスを標準化する傾向にあるため、将来的に C++ でヘテロジニアスをどのようにサポートできるかを示すことも目的です。以下に、その一例を紹介します。

重要な疑問 6: SYCL* の queue は ISO C++ に採用されるのか?

SYCL* は、queue を介してヘテロジニアス・デバイスにワークを割り当て、複雑なメモリーシステム (統一されコヒーレントであるとは限らない) 内のデータの受け渡しを行います。

queue クラスが C++ に採用されるかどうかを推測するのは時期尚早です。

C++23 の提案には、p2300 (英語) の「std::execution」など、特定のデバイスに実行を指示するさまざまな構文が含まれています。C++23 は今後も統合グローバル・メモリー・アドレス空間に依存し、リモートメモリー (複雑なメモリーシステム) はサポートしないことが分かっています。

構文にとらわれてしまいがちですが、いずれ、C++ が拡張されて完全なヘテロジニアスをサポートすれば、SYCL* の queue で具現化された概念が必要になります。それまでは、SYCL* がこの空白を埋めてくれます。並列ディレクティブやメッセージパッシングなど、いくつかの重要な機能は、独立した標準 (OpenMP* や MPI) にとどまっています。C++ が完全なヘテロジニアスをサポートしない可能性もありますが、いずれ段階的にサポートしていくと考えられます。

C++ は、新しい、証明されていない機能を実装する代わりに、確立されたベスト・プラクティスを標準化することを目指しており、SYCL* はそのための重要な足がかりの 1 つであると言えます。

C++23 が安定し、C++26 の検討に入ると、構文など、ヘテロジニアス・コンピューティング向けの将来の C++ の作成が始まり、完全なソリューションが完成するのは、さらに 5 ~ 10 年後になるでしょう。

SYCL* は、標準 C++ の中で、完全なヘテロジニアス・コンピューティングのプログラミングに対応するソリューションをすでに提供しています。これには、デバイスの列挙 (info)、ワークの定義 (kernels)、デバイスへのワークの送信 (queue)、リモートメモリーの管理などが含まれます。

重要な疑問 7: SYCL* の背後にいるのは誰か? 真にオープンなのか?

オープンで国際的な標準規格やオープンソース・ソフトウェア (OSS) プロジェクトは、業界全体に利点をもたらします。WiFi、USB、PCIe* から OpenMP*、MPI、Fortran、C、C++、OpenCL*、SYCL* まで、インテルと Codeplay は協力して、標準規格や OSS の開発と普及に取り組んでいます。

Apple は、低レベルの C インターフェイスである OpenCL* を推進する原動力となりました。SYCL* は元々、OpenCL* で C++ を使用したより高いレベルのインターフェイスを検討する取り組みから発展したものです。何年にもわたるオープンな議論の末に、SYCL* は誕生しました。Codeplay は、当初から SYCL* に貢献してきました。インテルは、FPGA 市場に参入し、インテル® Xᵉ アーキテクチャーを発表して GPU を計算処理に取り入れたことで、SYCL* への関心を高めていきました。インテルは、SYCL* 委員会の活発なメンバーであり、

SYCL* をサポートする実装に積極的に貢献しています。SYCL* はコミュニティーによる取り組みであり、この記事の著者 2 人もほかの多くの人々とともにコミュニティーに参加しています。

重要な疑問 8: なぜ SYCL* を信じるべきか?

複数のヘテロジニアス・マシン向けにアプリケーションをプログラムする必要がない方は、なぜ私たちが SYCL* の展望に興奮しているのか、理解しづらいかもしれません。SYCL* の必要性に疑問を持つのは、極めて論理的なことです。

ヘテロジニアス・プログラミングのユースケースはたくさんあります。CPPCON 2021 チュートリアル (英語) では、企業や国立研究所のプログラマーに、高スループットのワークロードを専用のアクセラレーターにオフロードする方法を紹介しました。

そのような多くの経験から、今後もヘテロジニアス・プラットフォーム向けの C++ プログラミングの必要性は高まり、SYCL* への関心も高まると確信しています。

ハードウェアの多様性の力を信じ、差し迫ったアーキテクチャーの爆発的な多様化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 (詳しくは、「C++ のヘテロジニアスな将来の考察」を参照してください)。

大規模な市場の実現にはオープンな業界標準が不可欠

新しい技術は独自開発として始まることが多く、ニッチなアプリケーションや市場を実現するにはそれで十分かもしれません。しかし、アプリケーションがテクノロジー・エコシステムに成長するにつれ、広く普及させるための競争と業界標準化の必要性が高まります。アクセラレーターを利用した計算処理は長年ニッチな機能でしたが、現在では一般的になりつつあります。これには複数の要因 (電力の壁、IPC の壁、メモリーの壁) が絡んでおり、そのすべてがなくなるわけではありません。

SYCL* や oneAPI など取り組みは、歴史的に独占的であったこの分野にオープンな業界標準をもたらすために登場しました。

最大の問題は、標準化への移行を熱心に推進する人と、独占的な利害関係に縛られている人がどれくらいいるかということです。

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るにつれ、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対する要求はさらに高まるでしょう。

SYCL* は、標準の策定とそれを実装するオープンソース・プロジェクトに、誰でも貢献できるオープンな標準です。コンピューター・アーキテクチャーの新しい黄金時代に、すべてのアクセラレーターにハイパフォーマンスへの道を明確に保証する、という目的を関係者全員が共有しています。

関連情報

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。

著者紹介

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。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)。

Codeplay Software の著名なエンジニアであり、現在 ISOCPP Foundation の理事兼副会長でもある Michael Wong 氏は、25 年以上の経験を持つ C++ 標準化委員会の上級委員で、C++ Directions Group のメンバーでもあります。WG21 SG19 Machine Learning および SG14 Games Development/Low Latency/Financials C++ グループの議長を務め、一般化属性、ユーザー定義リテラル、継承コンストラクター、弱順序メモリーモデル、明示変換演算子など多くの C++/OpenMP*/トランザクション・メモリー機能の共著者です。多数の研究論文を発表しており、C++11 に関する書籍の著者でもあります。また、数多くのカンファレンスで招待講演や基調講演を行っています。現在、SG1 Concurrency TS と SG5 Transactional Memory TS の編集者を務めており、カナダ標準評議会の SYCL* 規格とすべてのプログラミング言語の議長でもあります。以前は、OpenMP* の CEO として OpenMP* のアクセラレーター・サポートに携わり、IBM の XL C++ コンパイラー・チームを率いた後、IBM のコンパイラーを Clang/LLVM に移行させる技術戦略アーキテクトを務めていました。

---

[1] 「Elephants in the Room」とは、明白でありながら、少なくとも一部の人を不快にさせるため誰も言及しない重要な疑問と定義できます。