並列標準を洗練する：多様性、アライメント、相互作用

標準ベースの並列プログラミングの将来を見据える

John Pennycook インテル コーポレーション ソフトウェア・イネーブリング & 最適化アーキテクト

oneAPI は、言語（Khronos Group の SYCL*）、標準化されたライブラリー・インターフェイス（ニューラル・ネットワーク、データ・アナリティクス、その他）、およびハードウェアに近いプログラミング・インターフェイス（レベルゼロ）を含むアクセラレーター・プログラミングのオープン仕様（oneapi.io（英語）を参照）です。oneAPI は、ハードウェアとベンダーに依存しないように設計されており、複数のベンダーの CPU、GPU、FPGA、その他のアクセラレーターで動作することが実証されています。oneAPI は、プログラミングに標準ベースのアプローチを提供し、開発者がハードウェアとソフトウェア環境間の移植性を犠牲にすることなく高いパフォーマンスを実現できるようにします。

ソフトウェア・スタックのさまざまなレベルにわたる複数の**相互運用可能な**標準のコレクションが oneAPI を可能にしています。その一部（レベルゼロや SYCL* の拡張など）は oneAPI 仕様で採用されており、SYCL* や SPIR-V* などの業界標準は、oneAPI を使用した移植可能なプログラミングの基盤を提供します。さらに、OpenMP*、ISO C++、ISO Fortran などは、oneAPI でアクセラレーター・プログラミングの代替構文を記述するのに使用されます。

現在では、1 人の開発者が 1 つの言語を 1 つの抽象化レベルで使用して大規模なアプリケーションを作成することはまずないため、これらの標準間の相互運用性は非常に重要です。ほかの開発者が作成したライブラリーを呼び出して、さまざまな手法を組み合わせて使用する方がはるかに一般的です（例えば、SYCL* で作成された最適化ライブラリーを使用して Fortran や OpenMP* でアプリケーションを作成する、など）。異なる標準が新しい問題に対して独自のソリューションを提供することは珍しいことではないため、相互運用性を維持するには継続的な努力が必要です。標準が異なると、対象とする抽象化のレベルや推奨するコーディング・スタイルも異なることがあるため、新しい機能と既存の機能や確立された規約との関係を常に考慮する必要があります。しかし、標準は、互いに学習し合って、用語などを調整し、必要に応じて機能を互いに追加することがあります。

SYCL*、C++、OpenCL* の過去 10 年間を振り返ると、この調整と相互作用のプロセスが実際に作用していることが分かります（**図 1**）。例えば、OpenCL* 2.0 は C++11 のメモリーモデルの概念と用語を採用し、スコープと複数のアドレス空間の概念を拡張しました。その後、グループやグループ関数などの OpenCL* 2.0 の機能と C++17 の並列アルゴリズムを組み合わせた SYCL* 2020 のグループ・アルゴリズム・ライブラリーが作成され、開発者は使い慣れた C++ 構文を使用して、ハードウェア階層の複数のレベルでベンダーに最適化された操作にアクセスできるようになりました。

これは現在進行中のプロセスであり、IWOCL 2023 で発表した「SYCL* と ISO C++ の並列処理の調整に向けて」（英語）では、SYCL* のワークアイテムや C++17 のスレッド実行などの概念の間のギャップを埋めるように設計された、SYCL* に対するいくつかの新しい調整を提案しました。これは、特定の C++17 実行ポリシーが SYCL* デバイスで正しく動作できるようにするために必要なステップであり、これらの調整により、SYCL* で使用する C++17 並列アルゴリズムの動作の推測が容易になります。

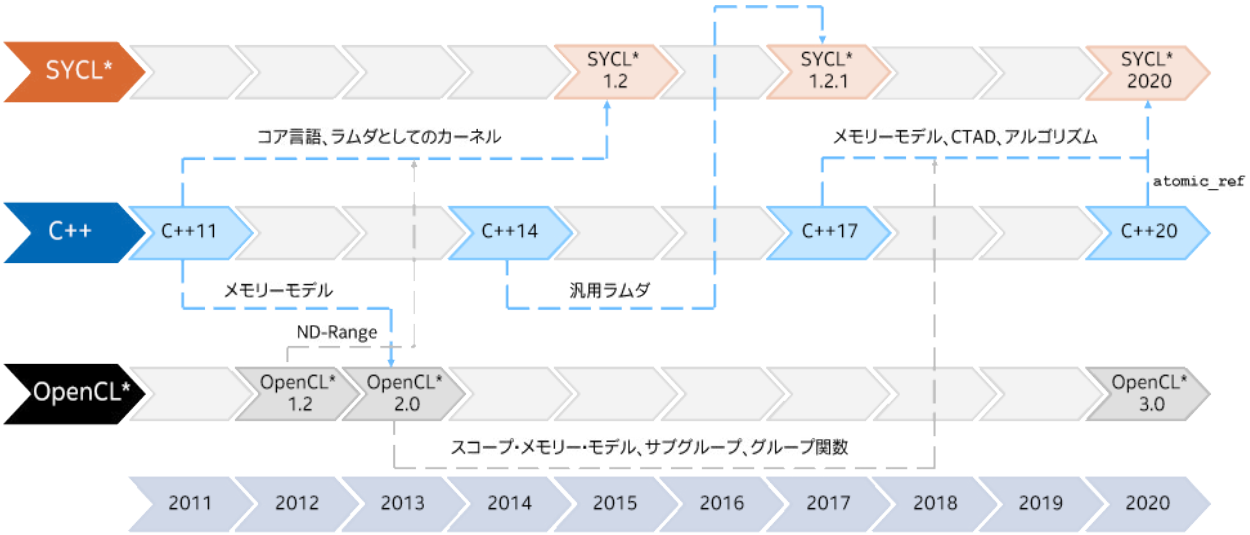

図 1. SYCL*、ISO C++、OpenCL* の並行進化

私たちは、SYCL* が C++ のヘテロジニアス・プログラミングの将来に影響を与える可能性があると考えています。そして、実装者の SYCL* での経験が、C++ 向けに提案されている新しい高レベルの抽象化の設計に反映されると信じています。また、SYCL* が今後も開発者にとって重要な役割を果たすと信じており、関連する C++ 抽象化と完全に相互運用可能なメカニズムを通じて、低レベルの最先端のハードウェア機能に対するダイレクトアクセスを提供します。

未来はどのようになるでしょうか？ 確かなことは言えませんが、P2300（英語）（`std::execution`）や P2500（英語）(「C++ 並列アルゴリズムと P2300」)など、いくつかの ISO C++ の提案にヒントが含まれています。これらの提案が承認されれば（早くても 2026 年まで実現しないでしょう）、**図 2** のようなコードを記述できるようになります。

```
// SYCL* を使用して特定の「scheduler」を表現
// get_scheduler() は(まだ)存在しないことに注意
auto q = sycl::queue{sycl::gpu_selector_v};
scheduler auto sch = q.get_scheduler();

// P2300/P2500 の機能を使用して並列ループを SYCL* スケジューラーに送信
std::for_each(std::execute_on(sch, std::execution::par_unseq), begin, end,
  [=](auto i) {
  ...
});
```

図 2. 未来を垣間見る - P2300/P2500 で提案されている機能を使用した C++ 並列アルゴリズムと SYCL* の組み合わせ

2026 年まで待てない人は、oneAPI DPC++ ライブラリー（oneDPL）を使用して C++ 並列アルゴリズムと SYCL* を組み合わせることができます。oneDPL は、C++ 並列アルゴリズムを SYCL* デバイスで実行できるようにするオープンソースのライブラリーであり、SYCL* が実行できる環境であればどこでも実行できるアプリケーションを開発する、生産性の高いソリューションを提供します。oneDPL は、ISO C++ の将来の拡張の可能性を検討する手段としても機能します。C++20 の Ranges に基づいた並列範囲ベースのアルゴリズム（英語）を標準化し、非同期並列アルゴリズム（英語）を表現する方法を検討する取り組みがすでに進行中です。oneDPL を使用して C++ 並列アルゴリズムを SYCL* デバイスに送信するのは、SYCL* 対応の実行ポリシーを使用するのと同じくらい簡単です（**図 3**）。oneDPL の試験的な機能を使用すると、遅延評価ビューを活用してカーネル融合を有効にし、さらに高いレベルのパフォーマンスと生産性を実現することができます（**図 4**）。

```
// oneDPL を使用して並列ループをデフォルトの SYCL* デバイスに送信
std::for_each(oneapi::dpl::execution::dpcpp_default, begin, end,
  [=](auto i) {
  ...
});
```

図 3. 現在 oneDPL で利用可能な機能を使用した C++ 並列アルゴリズムと SYCL* の組み合わせ

```
// 3 つのアルゴリズムを 3 つのカーネルとしてデフォルトの SYCL* デバイスに送信
using namespace oneapi::dpl;
reverse(execution::dpcpp_default, begin(data), end(data));
transform(execution::dpcpp_default, begin(data), end(data), begin(result),
          [](auto i){ return i * i; });
auto res = find_if(execution::dpcpp_default, begin(result), end(result),
                   pred);

// 範囲変換のパイプラインをデフォルトの SYCL* デバイスに送信
// 単一のカーネルとして実行される
using namespace oneapi::dpl::experimental::ranges;
auto res = find_if(execution::dpcpp_default,
                   views::all(sycl::buffer{data})
                   | views::reverse
                   | views::transform([](auto i){return i * i;}), pred);
```

図 4. oneDPL の試験的な機能を使用した C++ の範囲と SYCL* の組み合わせ

oneDPL の現在の構文は最終的な構文と多少異なる可能性がありますが、作業中の標準化プロセスではよくあることです。将来の C++ 標準として最終的に採用されるものは、複数のライブラリーと複数のハードウェア・ベンダーの経験を取り入れ、共通の要件とベスト・プラクティスを確立する長年の作業の結果となるでしょう。

開発者は OpenMP*、SYCL*、ISO C++、ISO Fortran のどれを使用してプログラムを作成するべきか質問されることがあります。答えは「どれでもかまわない」です。これらはすべて、ヘテロジニアス・プログラミングに対する有効なアプローチであり、それぞれ固有の長所と短所があります。Fortran の `ISO_C_BINDING`、OpenMP* の `interop` 節、SYCL* のバックエンド相互運用性インターフェイスなどの機能のおかげで、これらのアプローチを組み合わせることは、これまでになく容易になりました（**図 5**）。oneAPI の中核となる標準間の相互運用性に重点を置くことにより、当面の業務に最適なツールを組み合わせて使用することができます。

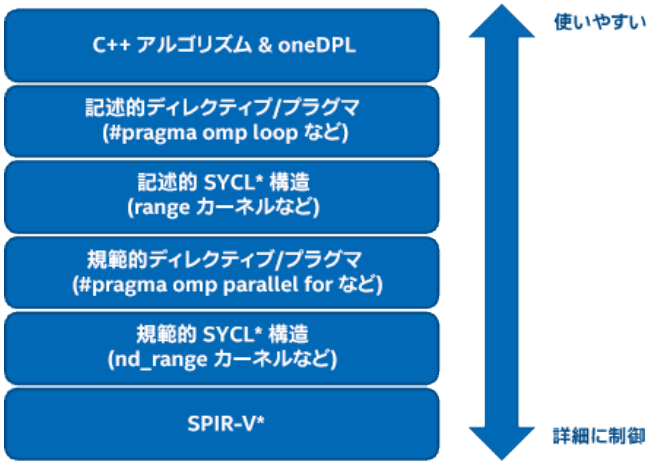

図 5：ヘテロジニアス・プログラミングに対するアプローチの比較：高いレベルの抽象化はハイパフォーマンスなプログラムを作成する生産性の高いソリューションを提供し、低いレベルの抽象化は熟練の開発者に詳細な制御を提供する