oneDPL による C++ プログラミングの生産性とパフォーマンスの向上

ヘテロジニアス・コンピューティングではオープンな標準と移植性が味方になる

Pablo Reble インテル コーポレーション ソフトウェア・エンジニア

プログラマーが新しいマルチコア・プロセッサーで直面する課題について、Herb Sutter 氏が「The Free Lunch Is Over（フリーランチは終わった）」（英語）と述べた 2005 年以来、我々は長い道のりを歩んできました。今日のコンピューティング・アーキテクチャーとアクセラレーターのポートフォリオは非常に豊富で、絶えず成長しています。半導体の根本的な制限と、強力で電力効率の良いコンピューティングに対する要望により、開発が推進されています。コンピューティングの世界は、ますますヘテロジニアスになり、プログラマーの課題を生み出しています。

C++ は、現在でも最も人気の高い 5 つのプログラミング言語の 1 つです（TIOBE（英語）のプログラミング言語ランキングでは、2022 年 1 月時点で第 4 位です）。メモリー管理の完全な制御や汎用プログラミングのサポートなどの属性により、ヘテロジニアス・プログラミングの課題に取り組むための優れた言語として評価されています。プログラミング言語を選択する際の一般的な懸念事項は、開発者の生産性とコード保守のコストです。幸いなことに、以前の調査では、並列ビルディング・ブロックと C++ アルゴリズムを組み合わせると生産性の向上が期待できることが示されています。例えば、特定のアーキテクチャーに共通の機能とパターン（リダクションなど）を最適化した組込み実装は、パフォーマンスと開発者の生産性の両方を向上させます。[1, 2]

業界最高レベルの oneDPL（英語）の実装は、オープンソースの LLVM プロジェクトに貢献しました。そのおかげで、マルチスレッドの世界における開発者の労力は大幅に削減されます。[1, 6]

進化したクラシック STL アルゴリズム
古いものと新しいものでコードを高速化

C++ 言語は進化しており、その標準テンプレート・ライブラリー（STL）も進化しています。例えば、5 年前に実行ポリシーがアルゴリズム・ライブラリーに追加されたことにより、既存の C++ コードでも最新のプロセッサーのユビキタスな並列処理の恩恵を受けることができます。oneDPL は、異なるベンダーが移植可能な方法で従来のアルゴリズムの高速化バージョンを実装できるようにする、進化した C++ STL と考えることができます。

oneDPL は、SYCL* を使用して C++ アルゴリズム・ライブラリーを実装します。

> 「SYCL*（「シクル」と読みます）は、ロイヤルティー・フリーのクロスプラットフォーム抽象化レイヤーであり、同じソースファイルに含まれるアプリケーションのホストコードとカーネルコードと標準の ISO C++ を使用して、ヘテロジニアス・プロセッサーに対応したコードを記述できます。」[3]

SYCL* には、ダイレクト・アクセラレーター・プログラミングの学習曲線があります。C++ はプログラマーにメモリー管理の完全な制御を提供しますが、ホストメモリーとデバイスメモリーの分離という概念はありません。oneDPL は、ホストとデバイス間でデータを共有するための移植可能な方法として、SYCL* のメモリー抽象化に依存しています。oneDPL のアルゴリズム関数はすぐに使えて、C++ プログラマーには馴染み深く、さまざまなアクセラレーター向けに最適化されています。これにより、学習曲線が平坦になり、コードのパフォーマンスと開発者の生産性が向上します。

[1] 「oneDPL ライブラリーは SYCL* の上に構築されているため、ネイティブ SYCL* コードよりも優れていることを目にするのは非常に興味深いことです。」Deakin ほか、2021（40 ページ）[2]

oneDPL の能力を説明する簡単な例を次に示します。

```
#include <oneapi/dpl/algorithm>
#include <oneapi/dpl/execution>
#include <oneapi/dpl/iterator>
#include <iostream>

int main()
{
   std::vector<int> data{1, 1, 1, 2, 1, 1, 1};

   auto policy = oneapi::dpl::execution::dpcpp_default;
   auto maxloc = oneapi::dpl::max_element(policy, data.cbegin(), data.cend());

   std::cout << "Run on "
             << policy.queue().get_device().template get_info<sycl::info::device::name>()
             << std::endl;
   std::cout << "Maximum value is at element " << oneapi::dpl::distance(data.cbegin(), maxloc) <<
std::endl;

   return 0;
}
```

この例は、一般的な `maxloc` リダクション（つまり、最大値を含むデータセット内の要素の特定）を実行ポリシーで指定したアクセラレーターにオフロードします。インクルードされたヘッダーは ISO C++ に準拠しています。`max_element` のブロッキング動作も同様です。データの移動は、この例では暗黙的に処理されます。つまり、計算がアクセラレーターにオフロードされた場合、ランタイムは SYCL* バッファーによりデータをラップして、ホストとデバイス間のデータ転送を自動的に処理します。プログラマーがホストとデバイス間のデータ転送を明示的に制御できるモードも存在します。

SYCL* での並列アルゴリズムの実装に加えて、oneDPL は、カスタム・イテレーターなど、デバイス・プログラミングに必須の拡張機能をサポートしています。これらの拡張機能は、異なるプラットフォーム間の相互運用性を確保するために、oneDPL 仕様（英語）に追加されました。[4]

次のステップ

将来を占う

現在開発中の、ISO C++ に完全に追加されていない、いくつかの強力で試験的な oneDPL の機能と、それらにアクセスする方法を見てみましょう。

- C++20 では、C++ STL アルゴリズムを使用する際の表現力を大幅に向上できる `Ranges` が追加されています。`Ranges` は、`Views` で複雑なデータ・アクセス・パターンをサポートすることにより、アルゴリズムの有用性を強化します。これはすべて、より少ないコード行で行われます。現在、ISO C++ の `Ranges` アルゴリズムは実行ポリシー（つまり、アクセラレーター）をサポートしていません。oneDPL は、選択したアルゴリズムの Ranges を有効にし、カスタム SYCL* ビューなどの拡張機能を提供して、デバイス・プログラミングを有効にします。[7]

- 従来の C++ アルゴリズムは、関数呼び出しのブロッキング動作を含め、明確に定義されています。しかし、計算をアクセラレーターにオフロードするときに、ホスト・プロセッサーをブロックすることが常に適切であるとは限りません。ホストとデバイスの実行とデータ転送のインターリーブを可能にするため、一連の非同期アルゴリズムが oneDPL に追加されました。それらの機能は C++ アルゴリズムに似ていますが、ブロッキング動作はありません。非ブロッキング動作を制御するため、結果の代わりに C++ の `future` 形式のオブジェクトを返します。[8]

ほかにもあります。自動デバイス選択[4] のような機能も将来のリリースで計画されています。最新情報は、GitHub*（英語）をフォローしてください。

まとめ

oneDPL は、CPU、GPU、FPGA、その他のアクセラレーターに、高いパフォーマンスと生産性を備えた C++ ビルディング・ブロックを提供します。オープンスタンダード・ベースで、その仕様により、異なるプラットフォーム間での相互運用性が保証されています。インテルのリファレンス実装は、許容範囲でライセンスされたオープンソース・プロジェクトです。[5]

参考資料（英語）

1. 並列リサーチカーネル
2. リダクション抽象化機能の解析
3. SYCL* プログラミング言語
4. oneDPL 仕様
5. oneAPI DPC++ ライブラリー
6. インテルの Parallel STL と C++17 並列アルゴリズムを使用してパフォーマンスを向上する方法
7. oneDPL 範囲ベース API のアルゴリズム
8. oneDPL 非同期 API のアルゴリズム

oneAPI と oneDPL を使用したプログラミングの詳細（英語）

- oneDPL を使用してクロスプラットフォーム・プログラミングの労力を減らしハイパフォーマンスな並列コードを実現する
- インテル® oneAPI ベース・トレーニング・モジュール