金融サービスのリスク計算における oneMKL 乱数ジェネレーター・デバイス API

Andrey Fedorov インテル コーポレーション マス・アルゴリズム・エンジニア
Gennady Fedorov インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア
Vladimir Polin インテル コーポレーション AI ソフトウェア・ソリューション・エンジニア
Robert Mueller-Albrecht インテル コーポレーション プロダクト・マーケティング・エンジニア

柔軟な並列計算能力の必要性

バーゼル銀行監督委員会（BCBS）（英語）は、金融リスク軽減策の必要性の高まりにより、計算需要が 4 ~ 20 倍に増加すると予測しています。これには、気候変動、サプライチェーンの混乱、地政学的な変化に関連する、資産の変動を深く理解するための高度な市場予測アルゴリズムとリスク・シミュレーションが含まれます。

銀行および金融サービス業界（FSI）のソフトウェア開発者は、GPU ベースのアクセラレーションを備えたハイパフォーマンス・コンピューティング（HPC）環境を利用して、リスク・シミュレーション・モデルを実装しています。これらのワークロードは広範にデプロイされるため、さまざまなハードウェア・プラットフォーム構成で実行できることが重要です。

Unified Acceleration (UXL) Foundation（英語）の取り組みが支持され SYCL*（英語）の人気が高まるまでは、ワークロードを新しい環境にデプロイするたびにコードを移植してリファクタリングする必要があり、大きな課題と非効率性に直面していました。

モンテカルロ法は金融分野でよく知られた手法であり、シミュレーション・シナリオのシードとして高性能の乱数生成に依存しています。特にエキゾチック・オプションでは、予想投資利益率（ROI）が複雑すぎて直接計算できないことが多いため、オプションの価格付けに使用されます。アメリカン・オプション・モデルとヨーロピアン・オプション・モデルは、さまざまなオプション投資結果の確率予測に広く使用されている 2 つのモンテカルロ・シミュレーション手法です。

2020 年に C++ with SYCL* ベースの oneAPI マス・カーネル・ライブラリー（oneMKL）インターフェイス（英語）が登場して以来、oneAPI 仕様のこのコンポーネントとインテル® oneAPI マス・カーネル・ライブラリー（英語）のオープンソース拡張機能により、乱数ジェネレーター（RNG）ルーチン向けのデータ並列 C++ インターフェイス（英語）が提供され、よく使用される擬似乱数、準乱数、連続分布と離散分布を備えた非決定性生成器が実装されています。

ほかの oneMKL 関数ドメインと同様に、oneMKL RNG には実装の一部として次のものが含まれます。

1. 手動オフロード機能（乱数ジェネレーター・ホスト API）
2. デバイス機能 — SYCL* カーネルから直接呼び出し可能な関数群（乱数ジェネレーター・デバイス・ルーチン（英語））。

この記事では、oneMKL 乱数ジェネレーター（RNG）デバイス API を使用して、インテル® データセンター GPU 上での計算パフォーマンスをホスト API 実装の 2 倍に大幅に向上する方法を示します。

ヨーロピアン・オプション価格付けモデルとアメリカンオプション価格付けモデルのホスト API とデバイス API の RNG 関数呼び出しのパフォーマンスを測定して比較します。

さらに、モンテカルロ・アルゴリズムを使用したアメリカンオプション価格付けモデルを例として使用し、金融ワークロードを独自の CUDA* コードから SYCLomatic（英語）（またはインテル® DPC++ 互換性ツール）を使用してオープンなマルチプラットフォーム SYCL* プログラミング・モデルに移行する方法を詳しく説明します。

この記事で紹介する結果は、インテル® データセンター GPU Max 1550 上で oneAPI 2024.0 リリースを使用して得られたものです。

RNG デバイス API の基本

デバイス・インターフェイスの主な目的は、SYCL* カーネルから呼び出せるようにすることです。送信時間はアプリケーションの全体的な実行時間に大きな影響を与えるため、パフォーマンスが大幅に向上します。グローバルメモリー転送のコストを排除し、素早く乱数を取得し、同じカーネル内で乱数を処理します。次に例を示します。

ホスト API

```
auto engine = oneapi::mkl::rng::mrg32k3a(*stream, 1ull);
oneapi::mkl::rng::generate(oneapi::mkl::rng::gaussian<double>(0.0, 1.0),
                           engine, n, d_samples);

sycl_queue.parallel_for(
    sycl::nd_range<3>(sycl::range<1>(n_groups) * sycl::range<1>(n_items),
                      sycl::range<1>(n_items)),
    [=](sycl::nd_item<1> item_1) {
        post_processing_kernel(d_samples);
    }).wait();
```

デバイス API

```
sycl_queue.parallel_for(
    sycl::nd_range<3>(sycl::range<1>(n_groups) * sycl::range<1>(n_items),
                      sycl::range<1>(n_items)),
    [=](sycl::nd_item<1> item_1) {
        oneapi::mkl::rng::device::mrg32k3a<1> engine_device(1ull, n);
        oneapi::mkl::rng::device::gaussian<double> distr(0.0, 1.0);
        double rng_val = oneapi::mkl::rng::device::generate(distr, engine_device);

        post_processing_kernel(rng_val);
    }).wait();
```

上記のホスト API コードシーケンスは、CPU から GPU への呼び出しを開始する 2 つのインスタンスを示しています。

1. 少なくとも 1 つの SYCL* カーネルを含む `generate()` 関数呼び出し
2. `nd_range` および `post_processing_kernel` を含む、オフロードされた `parallel_for` ループカーネル

デバイス API コードシーケンスでは、これらがすべて単一のオフロードされた `parallel_for` SYCL* カーネル内に含まれており、CPU と GPU 間のデータ交換が最小限になります。

そのため、デバイス API を使用すると、実装に必要なカーネルの数がホスト API よりも少なくなります。RNG デバイス API は、GitHub* にあるオープンソース・プロジェクト、oneMKL インターフェイス（英語）の一部としても利用できます。

アメリカン・モンテカルロとヨーロピアン・モンテカルロのワークロードを別々に考えてみましょう。

アメリカン・モンテカルロ・オプション価格付けモデル

このベンチマークをインテル® GPU 上で実行するため、NVIDIA* 開発者コードサンプル GitHub* リポジトリー（英語）からオリジナルコードを取得しました。次に、SYCLomatic オープンソース・プロジェクトを使用して、ネイティブ CUDA* GPU コードを SYCL* に移行しました。このツールはインテル® oneAPI ベース・ツールキットに含まれています。Apache* 2.0 ライセンスの下で GitHub* から入手することもできます。SYCLomatic を使用すると、オリジナルの CUDA* コードを SYCL* に移植できます。一般的な CUDA* コードの約 95% が自動的に SYCL* コードに移行されます。

GitHub* リポジトリー（英語）でアメリカン・モンテカルロ・オプション価格付けモデルのサンプル・ソースコードを SYCL* に移行する手順を確認し、SYCL* 対応のサンプル・ソースコード・プロジェクトを調べてください。

プロセスを完了するため、手動でコードの一部を変更して、ターゲット・アーキテクチャーで必要なパフォーマンス・レベルになるようにチューニングしました。

さらに、SYCLomatic の移行が完了した後、ホスト・インターフェイス呼び出しを追加しました。使用する SYCL* カーネルの数を減らすため、次のカーネル（`generate_paths_kernel`）へのデバイス呼び出しを追加しました。この方法では、乱数が生成されるとすぐに使用されるため、必要なカーネルの数が減り、乱数を保存するためのメモリーが不要になります。

デバイス API インターフェイス機能を適用すると、従来のホスト・インターフェイス呼び出しと比較してパフォーマンスが最大 2.13 倍向上します。アプリケーション全体では、最大 15% のパフォーマンス向上を達成できました。

図 1 は、計算したパスの数に応じたアメリカン・モンテカルロ・ベンチマーク結果のパフォーマンスのスケーラビリティーを示しています。**RNG と次のカーネルのスピードアップ**は、RNG ホスト・インターフェイス呼び出しと `generate_paths_kernel` を個別に使用したホスト API バージョンと、RNG デバイス API と `generate_paths_kernel` を組み合わせて使用したバージョンを比較したパフォーマンスの向上を示しています。**ベンチマーク全体のスピードアップ**は、RNG ホスト API を使用した場合と、RNG デバイス API を使用した場合を比較したベンチマーク全体のパフォーマンスの向上を示しています。

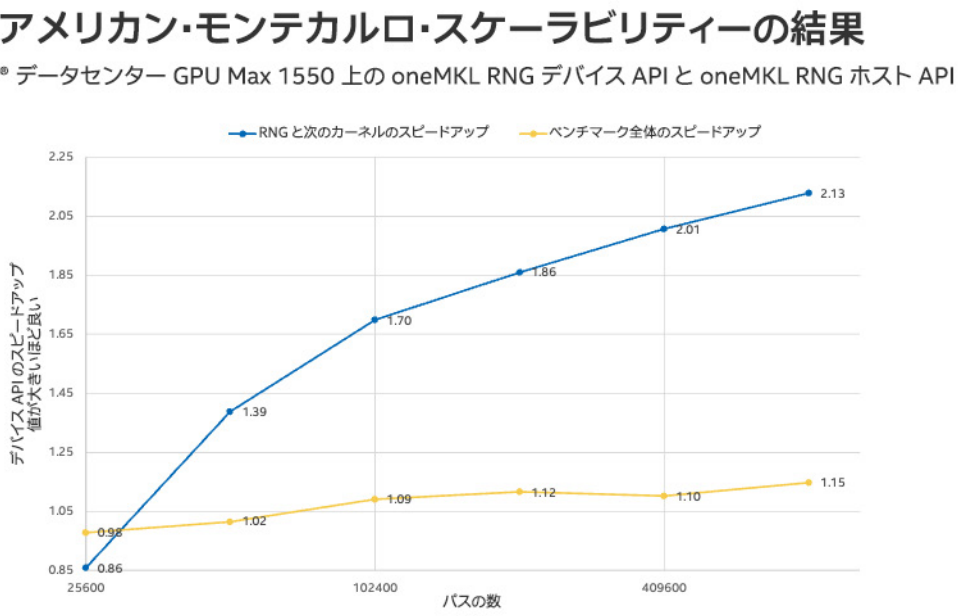

図 1. アメリカン・モンテカルロ・スケーラビリティーの結果

システム構成

テスト実施日：性能の測定結果は **2023 年 11 月 24 日現在のインテルの社内テストに基づいています。**
また、現在公開中のすべてのアップデートが適用されているとは限りません。
システム構成とワークロード設定：1 ノード、2x インテル® Xeon® Platinum 8480+ プロセッサー、56 コア、512GB。
Ubuntu* 22.04.2 LTS、カーネル : 5.15.47+prerelease70。インテル® oneAPI マス・カーネル・ライブラリー（インテル® oneMKL）2024.0。
インテル® DPC++ コンパイラー 2024.0.0（2024.0.0.20231017）。インテル® レベルゼロ、インテル® データセンター GPU Max 1550 1.3
[1.3.26690]。GPU : インテル® データセンター GPU Max 1550、GT 周波数 : 1.60GHz、1024 ユニット、2 タイル。

性能は、使用状況、構成、その他の要因によって異なります。

ヨーロピアン・モンテカルロ・オプション価格付けモデル

アメリカン・モンテカルロ・オプション・モデルのデバイス API とホスト API のパフォーマンスを確認したら、ヨーロピアン・モンテカルロ・オプション・モデルでも同じことを行います。oneAPI サンプル GitHub*（英語）で入手できるモンテカルロ・ヨーロピアン・オプション・サンプル（英語）のパフォーマンスを検討しましょう。

このユースケースでも、デバイス API を使用するメリットがあるか見てみましょう。

アメリカン・モンテカルロ・モデルとの違いは、リポジトリーのコードがすでにデバイス API を使用していることです。そのため、パフォーマンスを比較できるようにするには、ホスト API 実装を追加する必要があります。ホスト API と 1 年間のオプション価格付けモデルを追加するため、`option_years` 変数を `1` に設定します。この検討の後、デバイス呼び出しを別の SYCL* カーネルとしてのホスト呼び出しに簡単に置換できます。ただし、生成された数値を保存するメモリーが必要です。

これらの変更を適用すると、オリジナルのサンプルの構成で ~100 000 000 000 個の倍精度数を保存するには、GPU 上のメモリーが不足していることが分かりました。デバイス API を使用する欠点の 1 つは、CPU と比較して GPU では利用できるメモリーが限られているため、大規模なデータセットでは高度な共有メモリー管理が必要になることです。

デバイス API とホスト API の使用を除いて、リファレンス例を根本的に変更しないようにするため、計算する資産価格のパスの数を 16,000 に減らすことにしました。この変更により、コードを大幅に変更することなく、デバイス API 実装を使用してコードを正常に実行できるようになりました。

ヨーロピアン・オプション価格付け予測モデルのデバイス API とホスト API のパフォーマンスを比較すると、この場合も GPU デバイス API の使用により実行速度が向上していることが分かります。

図 2 は、RNG デバイス API を使用することにより、RNG ホスト API と比較してヨーロピアン・モンテカルロ・ベンチマーク全体のパフォーマンスが 3.69 倍向上する様子を示しています。

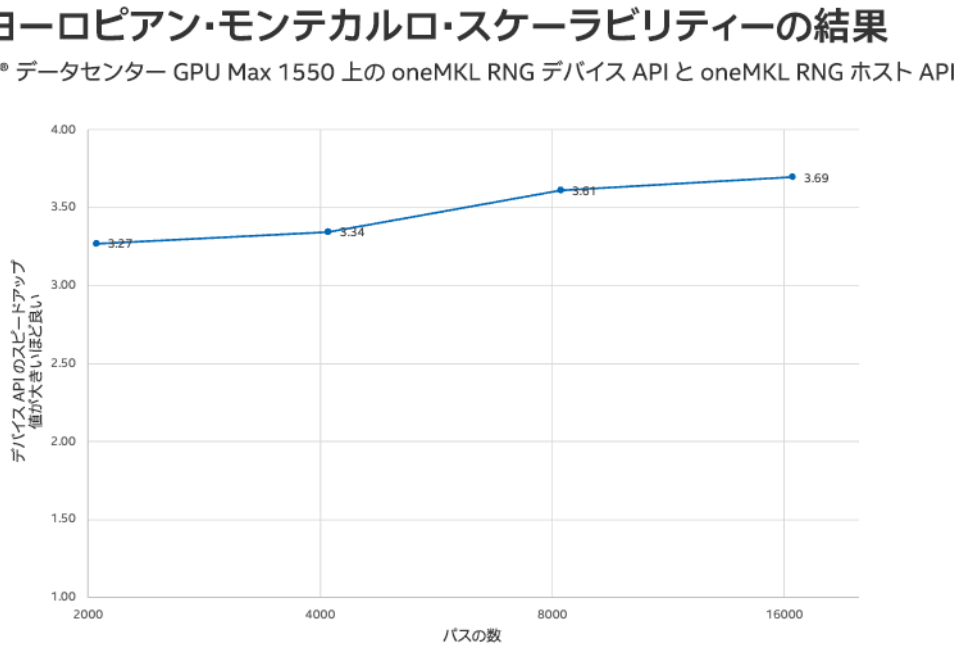

図 2. ヨーロピアン・モンテカルロ・スケーラビリティーの結果

システム構成

テスト実施日：性能の測定結果は **2023 年 11 月 24 日現在のインテルの社内テストに基づいています。**
また、現在公開中のすべてのアップデートが適用されているとは限りません。
システム構成とワークロード設定：1 ノード、2x インテル® Xeon® Platinum 8480+ プロセッサー、56 コア、512GB。
Ubuntu* 22.04.2 LTS、カーネル：5.15.47+prerelease70。インテル® oneAPI マス・カーネル・ライブラリー（インテル® oneMKL）2024.0。
インテル® DPC++ コンパイラー 2024.0.0（2024.0.0.20231017）。インテル® レベルゼロ、インテル® データセンター GPU Max 1550 1.3 [1.3.26690]。GPU：インテル® データセンター GPU Max 1550、GT 周波数：1.60GHz、1024 ユニット、2 タイル。

性能は、使用状況、構成、その他の要因によって異なります。

oneMKL RNG デバイス API ルーチンの活用

デバイス API を使用すると、ユーザーはホスト・インターフェイスと比較して本質的なパフォーマンス向上を達成できますが、特定のドメインの実装に関する多くのコーディングと知識が必要になります。デバイス API は、ユーザーが SYCL* 並列ルーチンのパラメーターを制御できる、柔軟な API でもあります。

この記事の結果は、重要なパフォーマンス・パスで乱数の計算を行う（金融以外の）ほかのアプリケーションにも適用できます。

課題に対処することは、インテル® コンパイラーと oneMKL ソフトウェア開発チームにとって日常的なプロセスです。インテル® ソフトウェアをユーザーにとって良いものにするため、我々は効率的なアルゴリズムと優れた最適化手法を常に模索しています。

この記事に興味を持たれた方は、まず、インテル® oneAPI マス・カーネル・ライブラリー、oneAPI マス・カーネル・ライブラリー（oneMKL）インターフェイス・プロジェクト (英語)、SYCLomatic（英語）を参照してみてください。

関連情報

- インテル® oneAPI マス・カーネル・ライブラリー
- oneAPI マス・カーネル・ライブラリー（oneMKL）インターフェイス（英語）
- SYCLomatic（英語）
- インテル® DPC++ 互換性ツール
- Unified Acceleration（UXL）Foundation（英語）