SYCL とインテル® oneMKL による定量金融の高速化

Andrey Fedorov インテル コーポレーション 数学アルゴリズム・エンジニア
Robert Mueller-Albrecht インテル コーポレーション テクニカル・プロダクト・マーケティング・エンジニア

金融市場では、さまざまな証券のリスクの分析、予測、評価に数学モデルを利用し、大規模データセットに適用しています。一般的な例としては、オプションなどのデリバティブ証券の価格決定や、ポートフォリオ管理におけるリスク管理が挙げられます。金融工学は、定量金融の数学的理論と計算シミュレーションを組み合わせて、価格決定、取引、ヘッジ、その他の投資判断を行います。

これらの計算には以下が使用されます。

- 微分方程式の数値解
- 多数のランダムシードを用いたモンテカルロ・シミュレーションに基づく証券価格決定シミュレーション

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

株式、株価指数、コモディティー、債券、通貨、転換社債契約、さらにはスワップや先物といったデリバティブなど、多くの金融商品はオプション市場に依存し、その影響を受けます。そのため、金融市場にとって、オプション価格決定の挙動を確実にモデル化し、ほかの資産クラスへの影響を適切に理解することが極めて重要です。

ギリシャ指標（リスク・パラメーター）

ギリシャ指標は、オプションやその他の資産クラスのリスク・プロファイルの評価基準となる、オプションに関連する一連のリスク・パラメーターであり、それぞれのリスクはギリシャ記号で表されます。追跡されるリスクは多岐にわたり、最も一般的なものは以下のとおりです。

- シータ（Theta）: 時間経過に対するオプション価格の減衰、または感応度
- ロー（Rho）: 金利変動に対するオプション価格の感応度
- デルタ（Delta）: 原資産価格の変動に対するオプション価格の感応度
- ガンマ（Gamma）: オプションのデルタと原資産価格の変動率に対するオプション価格の 2 次感応度
- ベガ（ニュー）（Vega（nu））: 原資産価格の変動率に対するオプション価格の感応度

ギリシャ指標の微分方程式の数値解は、平方根、指数関数、対数、正規乱数生成、相関乱数生成、パス生成など、さまざまな数学関数に依存します。

インテル® oneAPI マス・カーネル・ライブラリー（インテル® oneMKL）は、C、Fortran、SYCL API を用いたこれらの計算をサポートします。マルチアーキテクチャー GPU アクセラレーター・オフロードを活用するオープンソース、オープン・スタンダード・ベースの実装には、Unified Acceleration Foundations（UXL）のオープンソースの oneAPI マス・ライブラリー（oneMath）（英語）プロジェクトの SYCL API を使用できます。

シミュレーションと微分方程式の数値解を組み合わせた計算集約的な反復プロセスにより、金融資産のリスクモデリングは、SYCL のような高度な並列プログラミング・モデルの理想的な候補となります。

市場評価に用いられるさまざまな予測モデルと、インテル® データセンター GPU マックス・シリーズや最新世代のインテル® Core™ Ultra モバイル・プロセッサーの統合 GPU などの、高度に並列化された計算エンジン上で最適化された乱数生成器（RNG または RAND）を実行するメリットについて詳しく見ていきましょう。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

金融における乱数生成とシミュレーション

現代の定量金融において最も一般的なシミュレーション・モデルは以下のとおりです。

1. 二項式オプション価格モデル（BOPM）
2. ブラック - ショールズ・モデル（BSM）
3. ヘストンモデル
4. ヨーロピアン・モンテカルロ・オプション・モデル（EMC）
5. アメリカン・モンテカルロ・オプション・モデル（AMC）

それぞれに目的があります。

1. 二項式オプション価格モデル（BOPM）

このモデルは、一般化可能な数値手法を用いてオプションを評価します。このモデルは、時間の経過とともに変化する価格の「離散時間」（格子ベース）決定木を使用します。このアプローチは、オプションが満期日前であっても、いつでも権利行使できる場合に有効です。投資リスクは、将来の任意の時点において計算する必要があります。BOPM は、特定の時点ではなく、時間経過に伴う投資の変動に基づいているため、米国株式市場で取引されるオプションでよく使用されています。

二項式オプション価格決定は、ブラック - ショールズなどの他の代替手法よりも計算時間が長くなります。さらに、市場条件パラメーターの数が多い場合にはうまく機能しません。しかし、依存関係が単純なオプションでは、特に長期投資において正確な結果が得られるため頻繁に使用されます。

2. ブラック - ショールズ・モデル（BSM）

ブラック - ショールズは、株式や先物契約の価格が、一定のドリフトとボラティリティーを伴うランダムウォークに従う対数正規分布に従うと仮定します。この仮定により、ブラック - ショールズはオプション契約の価格決定に最適なモデルであると一般的に考えられています。ブラック - ショールズ方程式には、ボラティリティー、原資産価格、オプションの権利行使価格、オプションの満期までの期間、無リスク金利、そしてオプションの種類（コールまたはプット）の 6 つの変数が必要です。注意点としては、実際の市場では、BSM リスク分布予測よりも、高リスクの下落が頻繁に発生する傾向があることです。

3. ヘストンモデル

ヘストンモデルは、金融資産のボラティリティーの経時的変化を含めることで、BSM の欠点に対処します。このモデルでは、資産評価とボラティリティーは一定でも決定論的でもなく、確率的であると仮定します。ヘストンモデルの核となる偏微分方程式は、入力パラメーターを介した多数の連続的ランダムウォークのモンテカルロ・シミュレーションによって計算されます。ヘストンモデルは偏微分方程式であり、正確に解くことはできません。このモデルをオプションの価格決定に使用するには、数学的な近似との併用が必要です。このアプローチのアメリカンオプションとヨーロピアン・オプションのモデルへの適用については、次のセクションで説明します。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

4. ヨーロピアン・モンテカルロ・オプション・モデル（EMC）

モンテカルロ・シミュレーションは、さまざまな投資戦略の結果の特性を決定するために広く使用されている、繰り返しランダム・サンプリングに基づく手法です。ヨーロピアン・オプションでは、オプションは満期にのみ権利行使され、価格決定は、権利行使時までの固定期間に基づく、単純な金融ベンチマークです。

S_t を、次の式で表される確率過程に従う、特定の時点 t における株価とします。

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

ここで、μ はドリフト、σ はボラティリティーで、定数と仮定し、$W = (W_t)$ とします。$t \geq 0$ はランダム・サンプリング分布を表します。dt は時間ステップです。

$0 \leq t \leq T$ で定義されるヨーロピアン・オプションの価格 $V(t,S_t)$ は、原株の価格 S_t に依存します。オプションは $t = 0$ に発行され、満期 $t = T$ に権利行使されます。ヨーロピアン・オプションの場合、満期のオプションの価格 $V(T,S_T)$ は次のように定義されます。

コールオプション：

$$V(T,S_T) = max(S_T - X, 0)$$

プットオプション：

$$V(T,S_T) = max(X - S_T, 0)$$

ここで、X は権利行使価格です。問題は、価格 $V(0,S_0)$ を推定することです。

この問題に対するモンテカルロ・アプローチは、S_T の n 通りの実現可能性をシミュレーションし、$V(T,S_T)$ を平均化し、その平均を係数 e^{-rt} で割り引いて、オプションの現在の価格 $V(0,S_0)$ を算出します。S_T は対数正規分布に従います。

$$S_T = S_0 e^{(r-\sigma^2/2)T+\sigma\sqrt{T}\xi}$$

ここで、ξ は標準正規分布の確率変数です。

5. アメリカン・モンテカルロ・オプション・モデル（AMC）

アメリカン・オプション・モデルは、オプションの権利行使日が固定されていないため、やや複雑です。このオプションは、満期時または満期前にいつでも権利行使できます。これに対応するため、最小二乗モンテカルロ法を採用し、次の 2 つのステップを繰り返します。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

- まず、後方帰納法を実行し、すべてのタイムステップのすべての状態に再帰的に値を割り当てます。この値は、その時点のオプション価格の市場価格に対する最小二乗回帰として定義されます。この回帰のオプション価格は、(市場価格に依存する) 権利行使の可能性の値と、権利行使可能性のタイムステップ値の合計として定義されます。
- 次に、すべてのタイムステップのすべての状態を評価する際に、価格パスとその結果得られる状態の値に基づいて、各ステップでオプション行使に関する最適な決定を下すことでオプションの価格を計算します。この 2 番目のステップは、手順に確率的効果を加えるため、複数の価格パスで行うことができます。

アメリカン・モンテカルロ・オプション・モデルが、問題の並列実行パスの計算量を増やすことは明らかです。

計算負荷の高いタスクにおけるインテル® oneMKL SYCL API と GPU オフロードの利点を説明するため、これらの異なる手法の簡略化されたリファレンス・サンプル・ベンチマーク実装を見てみましょう。

インテル® oneMKL を用いた異なるオプション予測手法の実装とベンチマーク

金融オプションモデル間の考え方の違いは、実装における違いにつながります。

二項式、ブラック - ショールズ法、ヨーロピアン・モンテカルロ法では、初期データを生成する必要があります。oneAPI サンプルでは、インテル® oneMKL の乱数生成器 (英語) を用いて、各オプションの株価、権利行使価格、およびオプション年数の初期値を生成します。これはランダムな一様分布を用いて生成されます。この部分は通常計測されません。

```
...
  namespace mkl_rng = oneapi::mkl::rng;

  mkl_rng::philox4x32x10 engine(queue, rand_seed);

  sycl::event event_1 = mkl_rng::generate(
      mkl_rng::uniform<DATA_TYPE>(5.0, 50.0), engine, opt_n, h_stock_price);
  sycl::event event_2 = mkl_rng::generate(
      mkl_rng::uniform<DATA_TYPE>(10.0, 25.0), engine, opt_n, h_option_strike);
  sycl::event event_3 = mkl_rng::generate(
      mkl_rng::uniform<DATA_TYPE>(1.0, 5.0), engine, opt_n, h_option_years);
...
```

図 1. インテル® oneMKL を使用した初期乱数生成

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

二項式とブラック - ショールズ法の計算では、算術演算と、対数、平方根、指数などの標準的な数学関数のみを使用します。ベンチマークのカーネルには大規模な計算が含まれますが、各 SYCL ワークアイテムが 1 つのオプション（二項式）またはオプションの一部（ブラック - ショールズ）を処理するため、簡単に並列化できます。

```
queue->parallel_for(sycl::nd_range(...)),
  [=](sycl::nd_item<1> item) {
    for (.../* オプションの一部を処理 */) {
      const DATA_TYPE XexpRT = x * sycl::exp(-risk_free * t);
      DATA_TYPE n_d1 = DATA_TYPE(0.5) + DATA_TYPE(0.5) * sycl::erf(((sycl::log(s / x) +
(risk_free + DATA_TYPE(0.5) * sigma * sigma) * t) / (sigma * sycl::sqrt(t))) * sqrt1_2);
      DATA_TYPE n_d2 = DATA_TYPE(0.5) + DATA_TYPE(0.5) * sycl::erf(((sycl::log(s / x) +
(risk_free - DATA_TYPE(0.5) * sigma * sigma) * t) / (sigma * sycl::sqrt(t))) * sqrt1_2);
      const DATA_TYPE call_val = s * n_d1 - XexpRT * n_d2;
      const DATA_TYPE put_val = call_val + XexpRT - s;
      h_call_result_local[opt] = call_val;
      h_put_result_local[opt] = put_val;
    }
  }
);
```

図 2. ブラック - ショールズ・カーネルの擬似コード

ベンチマークの完全なコードは、ブラック - ショールズ･サンプル (英語) と二項式サンプル (英語) から入手できます。ヨーロピアン･モンテカルロ法では、株価、オプションの権利行使価格、オプション年数の初期値の生成に加えて、カーネル内部でランダムなガウス分布をその場で生成する必要があります。そのため、ベンチマークは二項式やブラック - ショールズよりも複雑になります。この生成には、以下の理由から、インテル® oneMKL 乱数生成器 (RNG) デバイス API を使用します。

- 生成されたデータを格納するメモリーは必要ありません。ホスト API を使用した場合、約 90GB のメモリーが必要になる可能性があります。
- ガウス分布のパラメーターは、処理中のオプションに依存します。
- インテル® oneMKL のホスト API を使用すると、API 呼び出し内でカーネル送信によるオーバーヘッドが発生します。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

```
namespace mkl_rng = oneapi::mkl::rng;
...
my_queue.parallel_for(
  /* sycl nd_range を設定*/,
  [=](sycl::nd_item<1> item)
  {
    /* オプション の一部を処理 */
    for(std::size_t i = 0; i < ITEMS_PER_WORK_ITEM; ++i)
    {
      ...
      mkl_rng::device::gaussian<DataType> distr(MuByT, VBySqrtT);

      /* 価格パスの一部を処理 */
      for (int block = 0; block < block_n; ++block)
      {
        auto rng_val_vec = mkl_rng::device::generate(distr, local_state);
        ...
      }
    }
  }
);
...
```

図 3. カーネル内での乱数生成

一部の乱数エンジン（例:MRG32k3a）では、コンストラクターの実行に時間がかかるため、あらかじめ別のカーネルでエンジンの状態を構築しておき、それらを生成処理とリダクションを含む後処理から構成される別のカーネル内で分布を得るために使うほうが効率的です。

```
namespace mkl_rng = oneapi::mkl::rng;
...

// 乱数エンジンの状態を初期化するカーネル
my_queue.parallel_for(
  sycl::range<1>(n_states),
  [=](sycl::item<1> idx) {
    auto id = idx[0];
    rng_states[id] = EngineType(/* エンジンのパラメーター */);
  }).wait_and_throw(); // 次のカーネルまでに処理を完了させる
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

```
...
// 計算された状態を使用して乱数を生成するカーネル
my_queue.parallel_for<k_MonteCarlo<DataType, ITEMS_PER_WORK_ITEM>>(
  /* sycl nd_range を設定 */,
  [=](sycl::nd_item<1> item)
  {
    auto local_state = rng_states[item.get_global_id()];

    for(std::size_t i = 0; i < ITEMS_PER_WORK_ITEM; ++i)
    {
      const std::size_t i_options = item.get_group_linear_id()*ITEMS_PER_WORK_ITEM+i;
      mkl_rng::device::gaussian<DataType> distr(MuByT, VBySqrtT);

      for (int block = 0; block < block_n; ++block)
      {
        auto rng_val_vec = mkl_rng::device::generate(distr, local_state);
        ...
      }
  }
);

...
```

図 4. 乱数エンジンの状態を個別に初期化

サンプルの最適化において、リダクションは興味深い部分です。生成された乱数の平均と標準偏差を計算する必要があります。ここでは、パフォーマンス上の理由から、各オプションの価格パスをワークグループの異なるワークアイテムに分割するため、SYCL のグループレデュースを呼び出す必要があります。もちろん、ほかのワークアイテムに依存しないように、すべての価格パスを 1 つのワークアイテムで処理するほうが簡単ですが、各オプションを 1 つのワークグループで処理し、価格パスをすべてのワークアイテムに均一に分散させるほうがパフォーマンスが向上することが判明したため、ここでは妥協案としてこのアプローチを採用しています。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

```
namespace mkl_rng = oneapi::mkl::rng;
...
my_queue.parallel_for(
  /* sycl nd_range を設定 */,
  [=](sycl::nd_item<1> item)
  {
    /* オプションの一部を処理 */
    for(std::size_t i = 0; i < ITEMS_PER_WORK_ITEM; ++i)
    {
      ...
      mkl_rng::device::gaussian<DataType> distr(MuByT, VBySqrtT);

      /* 価格パスの一部を処理 */
      for (int block = 0; block < block_n; ++block)
      {
        auto rng_val_vec = mkl_rng::device::generate(distr, local_state);
        auto rng_val = Y * sycl::exp2(rng_val_vec) - Z;
        for (int lane = 0; lane < VEC_SIZE; ++lane)
        {
          DataType rng_element = sycl::max(rng_val[lane], DataType{});

          // ワークアイテム内でレデュース
          v0 += rng_element;
          v1 += rng_element * rng_element;
        }
      }

      // ワークグループ内でレデュースして値を計算
      v0 = sycl::reduce_over_group(group_id, v0, std::plus<>());
      v1 = sycl::reduce_over_group(group_id, v1, std::plus<>());
      ...
    }
  }
);
...
```

図 5. 各オプションの全価格パスのリダクション

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

ベンチマークの完全なコードは、ヨーロピアン・モンテカルロ・サンプル（英語）から入手できます。

アメリカン・モンテカルロは複数のカーネルを含むため、上記のサンプルよりも複雑です。インテル® oneMKL の RNG ホスト API のガウス分布と MRG32k3a 乱数エンジンを使用します。パフォーマンス向上のため、RNG デバイス API を使用することもできます。

ベンチマークの完全なコードは、アメリカン・モンテカルロ・サンプル（英語）から入手できます。

GitHub リポジトリー（英語）にある手順に従って実行することで、インテルのソフトウェアおよびハードウェアでサンプルが正しい結果を生成することを確認できます。

インテル® oneMKL で金融リスク評価を高速化

オープンソースの oneMath とその SYCL API、そしてインテル® oneMKL を活用することで、迅速な結果と信頼性の高い金融資産リスク予測を実現できます。

インテル® oneMKL を使用すると、さまざまなプラットフォーム・アーキテクチャー構成で共通の API と SYCL を使用できます。

この記事の情報は、金融アプリケーションだけでなく、クリティカル・パフォーマンス・パスで乱数を計算するほかのアプリケーションにも適用できます。

インテル® コンパイラーおよびインテル® oneMKL ソフトウェア開発チームは、日常的に課題への対応に取り組んでおり、インテルのソフトウェアを改善するため、より効率的なアルゴリズムと最適化手法を常に模索しています。

ツールの使用を開始するには、インテル® oneAPI マス・カーネル・ライブラリー（インテル® oneMKL）、oneAPI Math Library（oneMath）（英語）、および SYCLomatic（英語）を参照してください。

そして、インテル® oneAPI ベース・ツールキットとインテル® HPC ツールキットをダウンロードしてください。

インテル® oneMKL、oneAPI、SYCL の関連情報

- インテル® oneAPI マス・カーネル・ライブラリー（インテル® oneMKL）
- oneAPI Math Library（oneMath）（英語）
- インテル® oneMKL 乱数生成器デバイスルーチン（英語）

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。

金融リスク計算の関連情報

- STAC-A2 : インテルの GPU を搭載した Dell PowerEdge サーバー上の oneAPI が金融市場のリスク分析を高速化（英語）
- 金融サービスのリスク計算における oneMKL 乱数生成器デバイス API
- Marco Dias : Real Options with Monte Carlo Simulation（英語）
- Rich Tanenbaum : Battle of the Pricing Models : Trees vs Monte Carlo（英語）
- Cox, J. C.; Ross, S. A. ; Rubinstein, M.（1979）. "Option pricing : A simplified approach（英語）". Journal of Financial Economics（英語）. **7**（3）: 229. DOI:10.1016/0304-405X（79）90015-1（英語）
- Rubinstein, M. (2000) . "On the Relation Between Binomial and Trinomial Option Pricing Models (英語) ," Research Program in Finance, Working Paper Series（英語）, Research Program in Finance, Institute for Business and Economic Research, UC Berkeley.

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
コンパイラーの最適化に関する詳細は、最適化に関する注意事項 (英語) を参照してください。