C++ for Finance : A multiple type Option Pricer

Benjamin Emily, Simon Carrière, Martin Verscheld, Hiba ElQoraichy

November 28, 2025

Introduction

Ce projet implémente un pricer multi-supports en C++ : modèles fermés Black–Scholes, Monte Carlo parallélisé (antithétiques, contrôle variate), et arbre binomial CRR pour gérer les options européennes, américaines et asiatiques via une interface commune d'options.

Organisation BinaryTree (h/tpp/cpp)

Interface (.h). Le fichier d'en-tête déclare le template `BinaryTree<T>` et inclut `BinaryTree.tpp` en fin de fichier pour que les définitions soient visibles dans chaque unité de traduction qui inclut l'interface.

Implémentation template (.tpp). Le `.tpp` contient les définitions des méthodes templées, séparées pour garder le `.h` concis tout en restant inclus par celui-ci (sinon on aurait des symboles non résolus au link).

Instanciations explicites (.cpp). Le `BinaryTree.cpp` inclut le couple `.hpp/.tpp` puis instancie explicitement les variantes réellement utilisées (`double`, `bool`, `int`) afin de centraliser la génération de code.

Séparer interface et implémentation template évite d'entasser toutes les définitions dans un unique `.h`, garde les headers lisibles, réduit les recompilations (les instanciations explicites sont centralisées dans un seul `.cpp`) et limite le risque de symboles dupliqués en link si un client oublie une `#include`.

Black Scholes Monte Carlo Pricer

A. Génération et parallélisme : `generate(nb_paths)`.

Principe général. Un appel à `generate(nb_paths)` ajoute *nb_paths nouvelles trajectoires* à l'estimateur courant (on peut appeler plusieurs fois pour accumuler).

Découpage multi-thread. On choisit

$$\texttt{thread\_count} = \min(\texttt{nb\_paths}, \texttt{hardware\_concurrency()}),$$

où `hardware_concurrency()` est le *nombre indicatif de cœurs matériels* selon la STL. Les `nb_paths` sont réparties en *chunks* quasi égaux (`base` + distribution du `remainder`). Chaque thread lance `simulate_chunk(paths)` et renvoie des statistiques locales (pas de partage d'état pendant la simulation, donc pas de verrou).

Simulation d'un chunk. On simule des paires *antithétiques*: à chaque pas, on tire $Z \sim \mathcal{N}(0,1)$ via `MT::rand_norm()` et on avance deux chemins en parallèle:

$$S \leftarrow S \times \exp(\text{drift} + \text{vol} \cdot Z) \quad \text{et} \quad S \leftarrow S \times \exp(\text{drift} - \text{vol} \cdot Z).$$

Antithétiques = technique de *réduction de variance*: coupler Z et $-Z$ annule une part de l'aléa. On remplit deux buffers `path_pos` et `path_neg` *locaux au thread* pour donner le chemin à `payoffPath`. Pour une européenne, seul le dernier point est utilisé par le payoff; pour une asiatique, toute la trajectoire est lue. Le payoff est actualisé par e^{-rT} (multiplicateur *d'actualisation*).

Les tirages pseudo-aléatoires proviennent d'un moteur Mersenne *thread_local*: chaque thread dispose de son générateur, éliminant toute contention ou data race sur le RNG.

Contrôle variate (vanilles). Si l'option est une vanille européenne, on active un contrôle variate parfait: on calcule une fois le prix Black–Scholes fermé et, pour chaque trajectoire, on remplace l'échantillon par `payoff - payoff + prix_BS` (variance quasi nulle et prix identique à la formule). Pour les options path-dépendantes ou américaines, le contrôle n'est pas utilisé et l'estimateur reste purement Monte Carlo.

B. Statistiques en ligne et agrégation.

Structure locale. Chaque thread retourne un triplet compact:

```
long long n; double mean; double M2;
```

- n : nombre d'échantillons produits par le thread,
- `mean` : moyenne en ligne (algorithme de **Welford**),
- `M2` : somme des carrés centrés, utile pour la variance $s^2 = M2/(n-1)$.

Welford (définition). Méthode *numériquement stable* pour mettre à jour moyenne et variance sans conserver tous les échantillons:

$$\mu_k = \mu_{k-1} + \frac{x_k - \mu_{k-1}}{k},$$
$$M2_k = M2_{k-1} + (x_k - \mu_{k-1})(x_k - \mu_k).$$

Fusion parallèle (Chan). On fusionne les agrégats d'un thread b dans l'agrégat global a via:

$$\mu \leftarrow \mu_a + \delta \frac{n_b}{n_a + n_b}, \qquad M2 \leftarrow M2_a + M2_b + \delta^2 \frac{n_a n_b}{n_a + n_b},$$

avec $\delta = \mu_b - \mu_a$. C'est *thread-safe* car chaque thread calcule d'abord ses statistiques *locales*, puis on agrège séquentiellement.

C. Différences.

- *Différences/plus* : parallélisation multi-threads, tirages `thread_local` (sécurisés en présence de threads), validation stricte des dates de monitoring, antithétiques et estimation en ligne sans stockage.