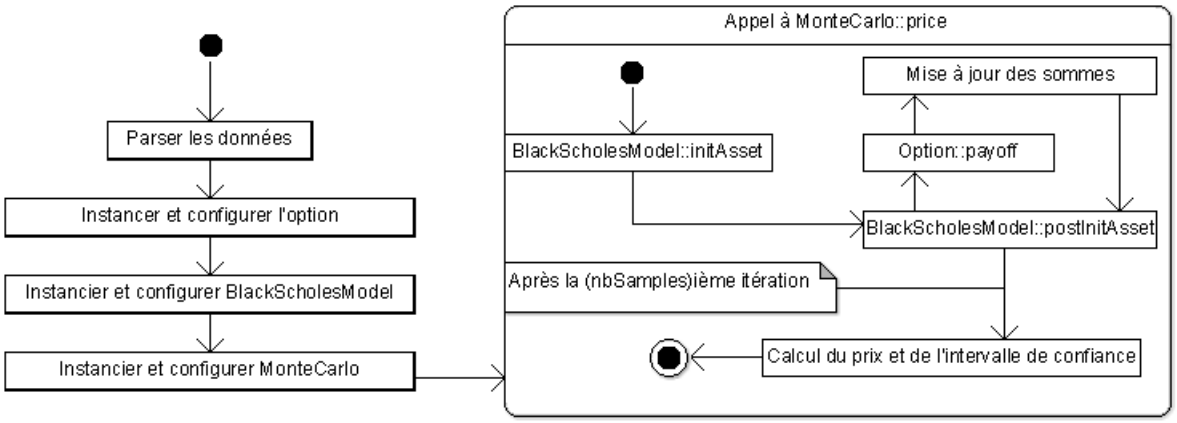
Grenoble INP - Ensimag
École nationale supérieure d'informatique et de mathématiques appliquées

Projet de Calcul Parallèle en Finance

Samy Amraoui - Julien Bernard

1 Question 1

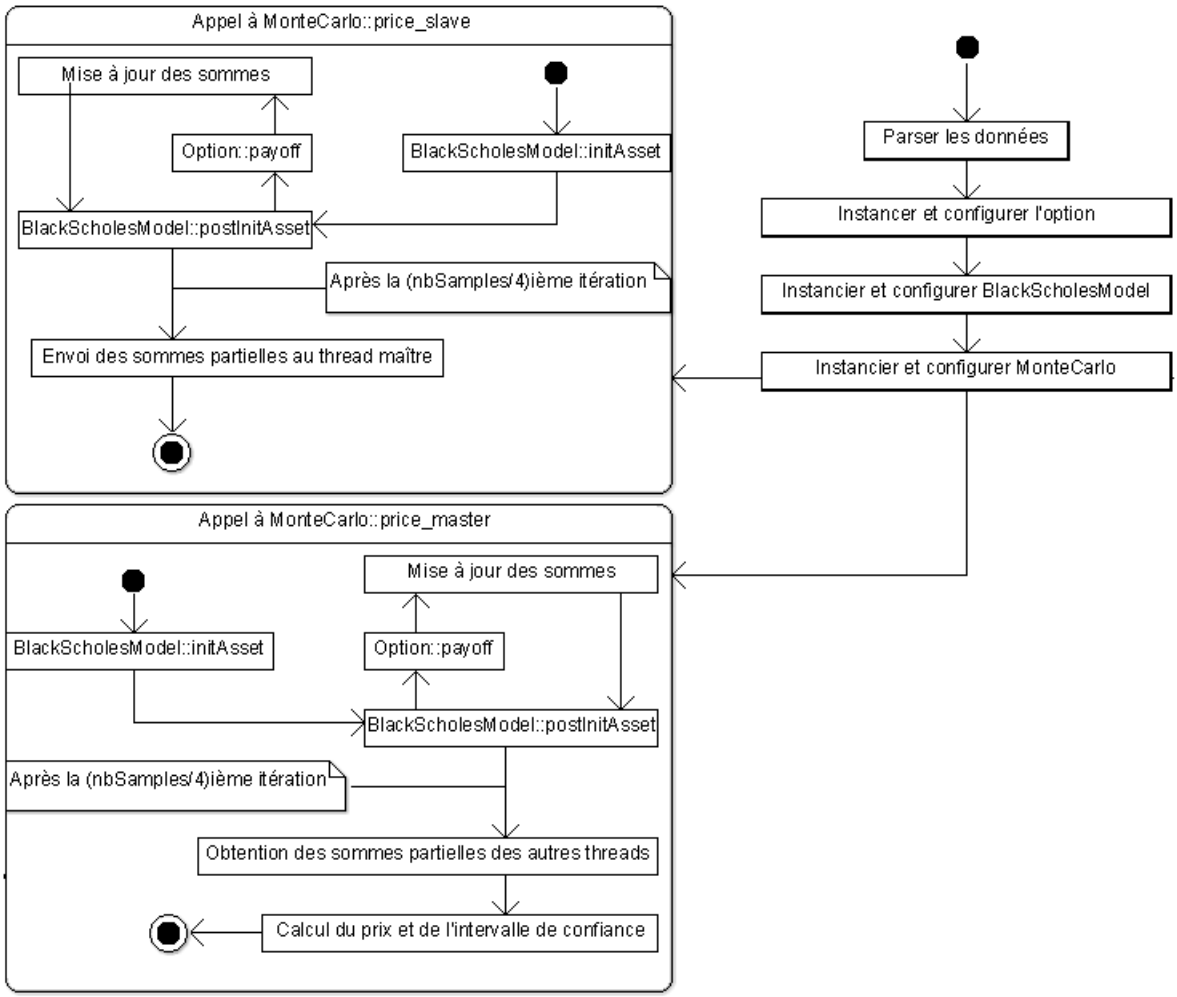


Les 2 fonctions métiers dans ce schéma sont BlackScholesModel : :postInitAsset et Option : :payoff. Il est donc nécessaire de paralléliser l'appel à ces dernières. Ces appels renvoient les données nécessaires à la construction des sommes (de payoff, et somme des carrés des payoff) qui permettent de déterminer le prix de l'option et l'intervalle de confiance associé.
En programmation MPI, donc en parallélisme à mémoires distribuées, ceci signifie que nous devrons envoyer les données à chaque appel à Option : :payoff, ou grouper les données localement avant de les communiquer pour les grouper globalement.
Par souci d'efficacité, la 2ème solution est de loin préférable.

Nous écrivons donc deux versions de MonteCarlo : :price, une pour le thread maître, et une pour le thread esclave. Chaque thread réalise (approximativement) autant d'appels à BlackScholesModel : :postInitAsset et Option : :payoff et construisent une somme partielle, suite à quoi les threads esclaves envoient cette somme partielle au thread maître, tandis que ce dernier les regroupe et calcule le prix et l'intervalle de confiance finaux.

2 Question 2

C'est ce que représente ce schéma :



3 Question 3

Pour la gestion des générateurs de nombres aléatoires en parallèles, on utilisera les générateurs Mersenne Twister paramétrisés pour l'indépendance, via la fonction pnl_rng_dcmt_create_id, à laquelle on passera comme id l'id du thread.

4 Question 4

La parallélisation a été faite à l'aide d'appels aux fonctions MPI_Bcast et MPI_Gather.

En modifiant le nombre de tirages des fichiers basket.dat et asian.dat à 1.000.000, on obtient des temps de calcul de :

Asian.dat :

— 41.84 secondes en séquentiel.

— 11.03 secondes en parallèle 4 threads.

Basket.dat :

— 9.4 secondes en séquentiel.

— 2.45 secondes en parallèle 4 threads.

Le speed-up observé est de $3,8 \pm 0,04$ pour 4 threads.

5 Question 5

On considère ici que l'entrée "précision" correspond à l'écart-type recherché.

L'écart type empirique d'une loi normale dont on connait l'espérance varie en $1 * n^{-}0.5$ avec le nombre de tirages.
La conclusion logique serait donc de tirer un échantillon, mesurer son écart-type, puis tirer (((écart-type recherché / écart-type actuel) $-$ 1)$*$ nombres de tirages effectués) tirages supplémentaires, de regrouper les informations suite à cela, de vérifier si la précision voulue est atteinte, et de recommencer si elle ne l'est pas.

Notons que le problème d'optimisation de la vitesse d'obtention du résultat en garantissant la précision avec cette méthode est beaucoup plus complexe mathématiquement que le raisonnement ci-dessus. Tout d'abord il la formule précédente n'est vraie que dans le cas où on connait l'espérance de la loi normale dont on estime l'écart-type. Ici nous estimons son espérance simultanément. La conséquence de ceci est que nous sous évaluons l'écart-type.

En effet l'espérance empirique est la valeur qui minimise la fonction $x \to \sum_1^{nbTirages}(x_i - x)^2$, tandis que le "vrai" x est l'espérance de la loi normale. Par conséquent, il serait nécessaire d'augmenter le nombre de tirages par rapport à la formule précédemment mentionnée pour estimer précisément le nombre de tirages à effectuer dans l'optique d'obtenir la précision voulue après une itération. Une étude mathématique poussée permettrait d'obtenir le nombre de tirages à réaliser pour obtenir une probabilitée donnée d'atteindre la précision après ces tirages.

Mais obtenir la précision voulue après une itération n'est pas nécessairement (ou plutôt, très probablement) pas la meilleure chose à faire. En effet la simulation de plusieurs trajectoires et autant de calculs de payoff peut être significativement plus couteuses temporellement qu'un transfert de 2 valeurs par thread. Cela dépend de la complexité de ces simulations, et du temps de transfert entre 2 threads. Il peut donc être préférable d'abaisser artificiellement le nombre de tirages à faire pour être sur de ne pas trop en faire, au prix de quelques transmissions. L'optimisation de ce problème nécessiterait une étude en profondeur.

Nous avons donc choisi de réduire de 10% le nombre de tirages obtenus par la formule précédentes, suite à quoi nous avons observé une accélération sur le calcul du prix de asian.dat avec précision 0.029 (5.03s contre 5.17s sur 10 appels au pricer).

Enfin, nous ajoutons également 40 tirages fixes, pour éviter les appels aux threads esclaves pour ne faire que quelques appels, qui ne sont pas rentables.

Nous obtenons alors les temps de calculs suivants :

Asian.dat, précision 0.01

— 293.51 secondes en 'parallèle monothread'.

— 77.36 secondes en parallèle 4 threads.

Basket.dat, précision 0.01

— 46.85 secondes en 'parallèle monothread'.

— 12.88 secondes en parallèle 4 threads.

Le speed-up observé est de $3,7 \pm 0,1$ pour 4 threads.