Finding the Right Level of Abstraction for Minimizing Operational Expenditure

MAXELER Technologies

This is a summary of a presentation at Workshop on High Performance Computational Finance at SC11, November 2011.

About the Authors

Erik Vynckier is with Scottish Widows Investment Partnership. James Spooner is VP of Trading Solutions, Stephen Girdlestone is a Senior HPC Engineer, Oliver Charlesworth is a Senior Applications Acceleration Engineer and Oskar Mencer is CEO of Maxeler Technologies.

Summary

Financial businesses have a strong connection between performance/efficiency of computation and success of the business. Computation sets the speed of trading and determines the degree of risk taken on by the business. The more scenarios that can be computed, the more combinations of future events that can be hedged against. This article proposes the possibility of a positive correlation between revenues and computational capacity.

Determining the total cost of ownership (TCO) of a technology is essential to fully comprehending the impact of that technology on a business. The main components of TCO include cost of software development, capital expenditure, operational expenditure, and indirect costs such as the cost of failures. In the financial industry, substantial business costs can originate from failed or slow computations. The total cost of ownership is usually not apparent to the programmer deciding program structure, level of abstraction, and time spent on performance optimization.

The application considered in the test case is a Monte Carlo single-asset option pricer, based on Heston's stochastic-volatility model extended to include multiplicative price jumps. A number of distributions are sampled, including Gaussian and Poisson; these are all derived from a simple multiply-with-carry uniform RNG.

Static analysis facilitates understanding of the structure and control flow of the source code. When optimizing an application on any platform we focus on loop structure and data flow. A loop graph is used to assist visualization of the code, shown in Figure 1. Ellipses represent blocks of computation, boxes represent loops and arrows between ellipses show the data flow. When static analysis is not feasible, we begin dynamic analysis. A tool called a Maxspot is used to create a visual of the representation. Maxspot records every memory access and control flow transition in a log to analyze the location of the control loops and the function of each block. Maxspot assigns importance to each block based on its function in the code and creates a corresponding graph.

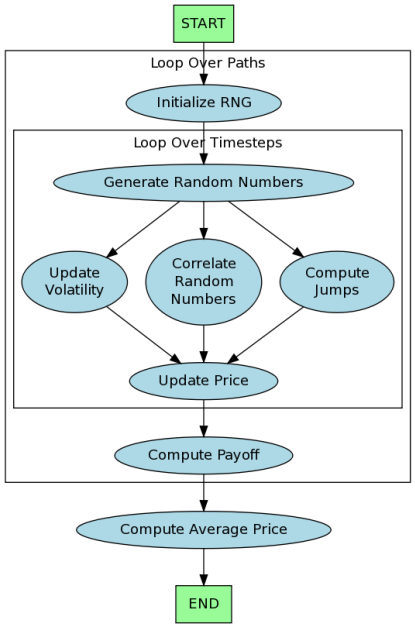

Figure 1: Static loop graph for Monte Carlo case study

The Monte Carlo application can be easily parallelized because each path is computed independently. Maxeler hardware accelerators implement a dataflow model where computations are described structurally as opposed to sequentially. A graph consisting of nodes is constructed where each node has a specific function. The output of one node serves as the input to the next node without needing to be written to memory.

The original Monte Carlo code was written in C, but it was rewritten in C++ to facilitate the use of additional nodes and payoff calculations. Unfortunately, ease of programming results in extra abstraction causing more function calls which ultimately slows down the application by 15%. If this application is run continuously this extra cost of operation is paid over and over again. Maxeler DFEs optimize in the opposite manner, requiring more programming effort but significantly reducing runtime.