Optimized VaR Computations Using High Performance Computing

Amit Kalele and Easwar Subramanian
TCS Innovation Labs, Tata Consultancy Services, India
kalele.amit@tcs.com, easwar@atc.tcs.com

Abstract

We consider the problem of computing value-at-risk (VaR) for a portfolio of financial contracts using multi-core CPUs and GP-GPUs. One objective is to outline optimal algorithms for computation of VaR using traditional multi-core systems and modern GP-GPUs that uses the respective compute resources efficiently. Further we demonstrate that the speed up achieved using GPUs for VaR computation is far higher in comparison to a traditional CPU-based parallel processing system. Since the GPU infrastructure also possess lesser energy requirements, the task is accomplished in a greener fashion as well.

Introduction

What is Value-at-Risk ?

Value-at-risk (VaR) is a financial risk measure used to quantify the likely loss in a mark-to-market value of a portfolio over a certain time horizon at a certain probability or confidence level. More precisely, for a given portfolio, confidence level and time horizon, VaR is the threshold value such that the probability that the losses to the current value of the portfolio over the given time horizon exceeds VaR is given by the assumed probability level.

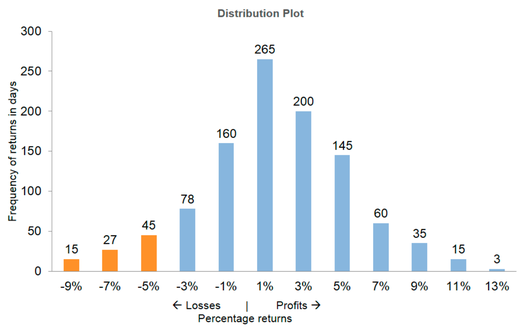

Figure 1: Distribution of portfolio returns over a certain time horizon. VaR is computed for a desired confidence level such as 95 % or 99 %. For example, the 95 % and 99 % VaR in the above figure corresponds to 45 and 15 units respectively.

Main Objective

The goal is to compute VaR of a large portfolio with thousands of contracts or investments using Multi-core CPU system and a GP-GPU system and compare their performance. Specifically, we compute historical VaR (HVaR) of a portfolio in which future scenarios of risk factors are generated based on their historical values. For more on VaR and approaches to calculate VaR, refer [1, 2].

On the need for HPC for VaR computation

- Re-evaluation of portfolio for many possible future scenarios.
- Portfolio size can be large.
- Portfolio valuation may require Monte-Carlo simulations.

Benefits in using HPC capabilities are :

- Speed up that can be obtained as compared to the standalone systems for portfolios of large sizes.
- The second is the possibility to obtain accurate estimates for VaR if the portfolio contains exotic contracts which can only be priced using Monte-Carlo techniques.

Our Setup

Nature of Portfolio : Our portfolio consists of simple European options, numbering up-to 16K. Pricing is formula based. Goal is to devise architecture specific HVaR computation schemes and compare performance.

Multi-core CPU system : Intel clover-town four core 3 GHz CPU with 24GB RAM. Sequential implementation is done on a single core while the multi-core version is executed on all four cores.

GP-GPU configuration : Nvidia Tesla C2075 containing 14 streaming processors each having 32 cores and 5GB GPU RAM

Generic HVaR Computation Algorithm

Following steps enumerate the various steps in HVaR computation.

1. Identify suitable risk factors for the portfolio and retrieve historical values of risk factors for a pre-defined time period.
2. Generate possible generate future scenarios of risk factors based on historical values
3. Re-valuate each contract in the portfolio for every possible future scenario .
4. Contract values for a particular scenario are summed to obtain a portfolio value for that scenario
5. Evaluate profit and loss (P&L) numbers by comparing current portfolio value and portfolio values obtained for each scenario
6. Compute portfolio VaR from P&L values at the desired confidence level

The most time consuming step in the historical VaR calculation is pricing of the portfolio for every future scenario. The value of a call option with strike K and time to maturity T – t, in a market with risk-free interest rate r, written on an underlying with spot price S and volatility σ and paying no dividend (d = 0), at an arbitrary time t < T is given by,

$$C(S,t) = N(d_1)S - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$d_2 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$= d_1 - \sigma\sqrt{T-t}$$

where C(S, t) is the price of the call option at time t and N(.) denotes the cumulative normal distribution.

Multiple ways to parallelize the solution :

1. A compute unit can be made to compute the value of one contract for all future scenarios.
2. A compute unit can price all contracts of a portfolio for a single scenario.
3. A compute unit can price one contract for one scenario.

Multi Core CPU Implementation

We use up-to four cores to compute the HVaR. The parallel implementation is done with OpenMP protocol. In case of using option 2 listed above, the four CPU cores would simultaneously compute the price of the portfolio for a single scenario. That is, if there are N contracts in the portfolio, each CPU core computes the price of N=4 contracts for a given scenario.

GP GPU Implementation

1. **Approach to GPU Parallelization** - In the current implementation, there are 261 scenarios (corresponding to one year) and N contracts amounting to 261* N pricing computations. We spawn equal number of threads (option 3); each computing price of an instrument for a scenario. The adopted choice of parallelization not only achieves highest degree of parallelism but also reduces the number of computation per thread to minimum.
2. **Global memory access pattern** - The single most important parameter in GPU optimization is global memory access pattern. Ideally consecutive threads should access adjacent elements from global memory of GPUs. The data structure used to store all contract with parameters (r, d, sigma, K, S) was an array of structure (AoS). This leads to strided access. Remapping this data structure to separate arrays for each parameter enables coalesced access of global memory.

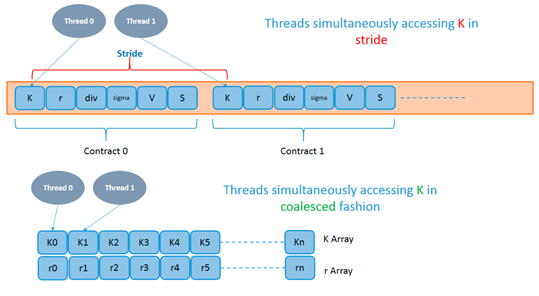

Figure 2: Global Memory Coalesced Access.

3. **CPU and GPU overlap** - CPUs are usually free once they offload computations to GPUs which can be utilized to speed up computations further. The aggregation step (adding individual contract prices to compute portfolio price) is a reduction operation. This requires two stages in GPUs. In first stage, N instruments are grouped into blocks and sum of these individual blocks (partial sum) is computed. The addition of partial sums (2nd stage) is carried out on CPU. This allows us to free the GPU cores for another batch of computation while CPU is performing 2nd stage of aggregation. This way we achieve an overlap between CPU and GPU computation.
4. **Multiple stream computation** - GPUs now support multiple stream computation and different kernels can be executed on GPUs at the same time in different streams. This allows us to have data transfer-compute overlap. The number of contracts are divided into 4 batches and each batch is processed in a different stream. While computation for 1st batch is underway, the data for next is transferred in parallel. This allows overlap and reduces wait time for GPU cores.

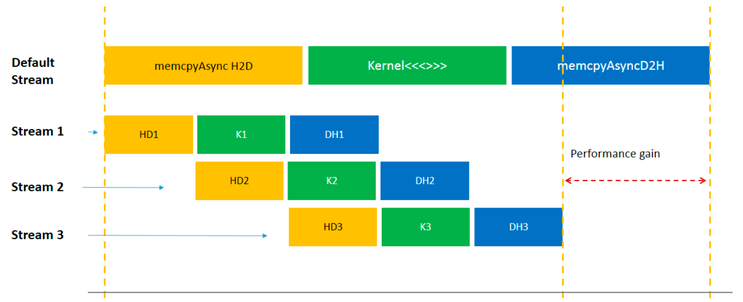

Figure 3:Computing with Multiple Streams

5. **Local Registers** - All the frequently required variables, which remain constant over the period of execution are copied to local registers. This reduces frequent global memory access.

Performance Comparison

The performance results of sequential, multi-core CPU and GPU versions are graphically depicted below. The performance gain for multi-core CPU with respect to the sequential implementation is about 3X (for 16K instruments) which is bounded by the number of CPU cores available. On the other hand, the GP-GPU performance is about 53X (for 16K instruments) faster than the sequential version and 17X faster than the multi-core CPU implementation.

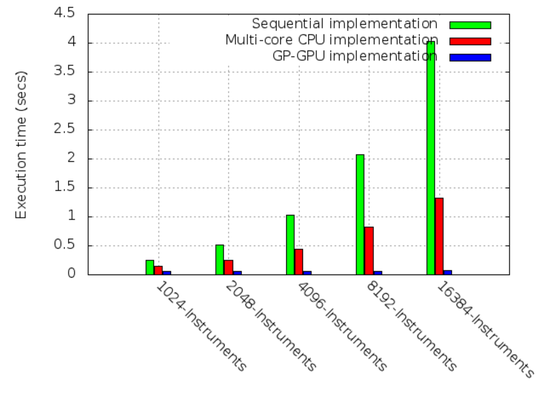

Figure 4: Performance comparison for sequential and parallel implementations

References

[1] P. Jorion Value-at-Risk: the new benchmark for managing financial risk, 3rd edition, McGraw Hill, 2007.
[2] J. C. Hull. Options, Futures and other derivatives, 7th edition, Prentice Hall, 2008.