White Paper

High Performance Computing
Financial Services

# More Than a Thousand-fold Speed-up for xVA Pricing Calculations with Intel® Xeon® Scalable Processors

## Significant performance leaps, from Matlogica technology and 2nd Generation Intel Xeon Scalable processors, demonstrate upside from Intel® AVX-512, CPU memory capacity, and multithreading or multiprocessing.


**Author**

**James Reinders**

HPC Enthusiast

James Reinders Consulting




**Table of Contents**



Substantial performance improvements are possible for xVA[1] Pricing and Greeks applications, using Intel Xeon Scalable Processors along with innovative easy-to-integrate software from Matlogica.

Since the 2008 financial crisis, financial institutions have been forced to report their sensitivities to multiple scenarios, requiring banks to recalculate their trade portfolios thousands of times. Regulations push banks to calculate risks to even finer risk granularities, making regulatory adherence an even larger burden in terms of costs for implementation, running, and maintenance.

Sophistication in financial market risk modeling gave rise to multiple "Valuation Adjustments" such as funding (FVA), credit risk (CVA), regulatory capital costs (KVA), and more—hence the "*x*" in *x*VA to denote an entire collection of valuation adjustments to meet individual needs. These values represent price correction at a portfolio level and therefore can only be computed once price distribution of each trade over portfolio lifetime is known. This means thousands of price valuations are needed for each trade to accurately determine CVA and Debit Valuation Adjustment (DVA) values. Adding to the challenges, computing fine grade sensitivities for these adjustments using traditional bump-and-revalue methods has become computationally infeasible or they demand a hefty cloud computing cost. For this reason, utilizing Automatic Adjoint Differentiation (AAD)[3] becomes a highly desirable method to implement robust risk management in current markets and comply with current and future regulatory requirements.

More efficient methods to support xVA computations can yield important benefits for a wide range of business groups including sales, trading, risk management, product management, enterprise risk, and treasury.

*One xVA Pricing application was accelerated by as much as 1770X*

Results published by Matlogica have demonstrated outstanding performance for financial derivative computations. One xVA Pricing application was accelerated by as much as 1770X. Workloads that included xVA Pricing and Greeks[4] together demonstrated up to 832X performance gains even when compared with another solution that also utilized AAD. These results utilize innovative patent pending technology from Matlogica that takes advantage of the advanced capabilities in Intel Xeon Scalable Processors.

## The need for differentiable programs

There are plenty of numerical computing problems that require not only processing of input values into output values, but also require measures of how

outputs change with respect to small variations in inputs. These measures are called gradients, derivatives, sensitivity, or risk in finance applications.

Examples of such problems include machine learning (ML) and deep neural networks (DNN) training, but also include reconstruction of the initial model state for weather modeling, shape optimization problems in industrial design in conjunction with computer fluid dynamics, and real-time scanning and reconstruction of deep underground oil and gas reserves.

Applications in Finance are wide and include computation of risk sensitivities of financial derivative instruments, calibration of model parameters, and pricing.

The finance world has traditionally relied on the "bump-and-revalue" method that applies small change to input data, re-runs analytics, and approximates sensitivities as ratios of output value change with respect to input data change. Complexity of this process grows linearly with the number of input variables.

In cases where the number of output variables is low, but the number of inputs is high, the Adjoint Differentiation (AD) method can help. Roughly speaking, the AD method passes all calculations from the output back to all inputs and accumulates derivatives according to the chain rule. Complexity of the AD method grows linearly with the number of output variables, which for many problems are just a few values (e.g., loss function value for ML/DNN training, portfolio value). Since computational costs do not depend on the number of input variables, one application of the AD method to original analytics computes sensitivities for all inputs. This is what makes AD so appealing to practitioners and allows them to obtain results faster and save on compute costs.

The method can be manually implemented using chain differentiation from high school math, starting from the output of the original program and propagating differentials all the way to program inputs. This process is known as back propagation in DNN, but also used widely in other applications and known as adjoint (or reverse) program of the original.

Manual AD implementation is tedious and must be consistent with the original (forward) program. Implementation and maintenance of adjoint and the original versions for constantly evolving programs is even harder. Therefore, Automatic AD (AAD) tools exist to simplify this step generating the reverse program automatically.

## New programming paradigm yields tremendous performance returns

Matlogica is a UK-based company specializing in software solutions that help accelerate Monte-Carlo Simulations using highly parallel vectorized software and automatic adjoint differentiation. Their ground-breaking results come from bringing together an impressive range of specialists including quantitative analysts, computer science engineers, and researchers.

The approach Matlogica takes to acceleration is novel in both its easy-to-use programming interface and the high performance it achieves out-of-the-box. Straight-forward and minimal code changes, to make use of the libraries, offer leaps in performance for both xVA Pricing and Greeks calculations.

To accomplish their published results, Matlogica utilized two complementary products. The Matlogica Vector Accelerator C++ Library enables high-performance processing of data using SIMD, multithreading, and multiprocessing capabilities of modern processors. The Malogica Parallel AAD-C[5] C++ Library implements automatic adjoint differentiation of C++ programs with focus on computations where derivatives are required for multiple instances of input data.

## Outstanding performance that is easy to add into preexisting applications

To illustrate these achievements, Matlogica assembled an xVA benchmark modeling one interest rate curve with four

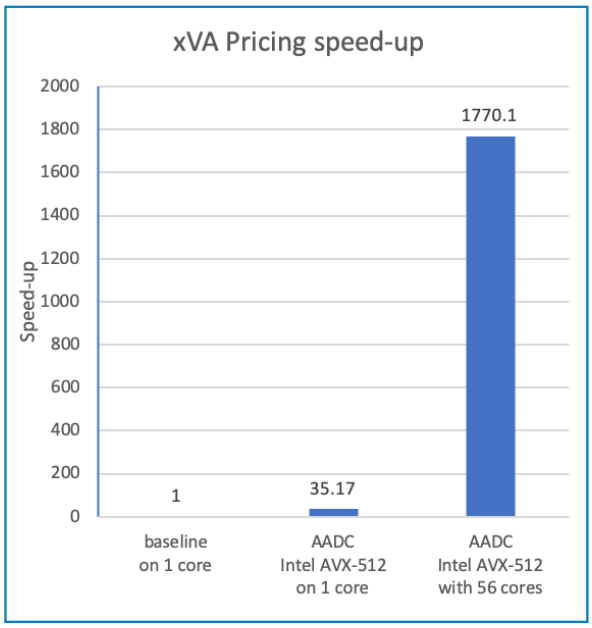


**Figure 1.** xVA pricing speed-up

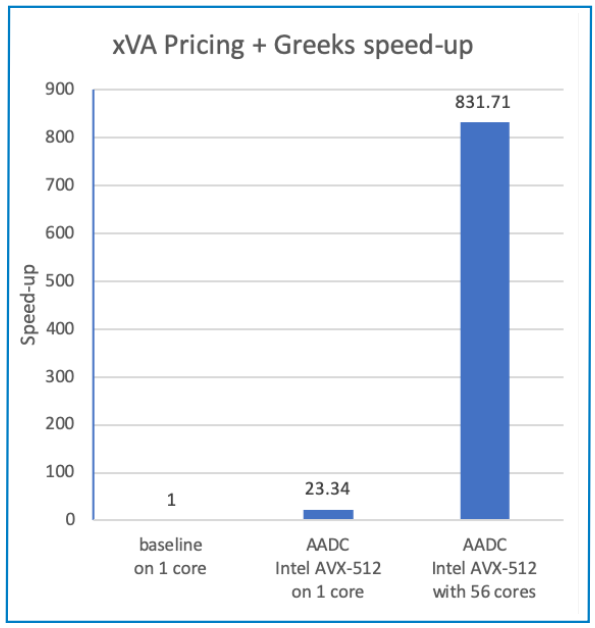


**Figure 2.** xVA pricing + Greeks speed-up

projection curves, with IR swaps generated randomly. It computes CVA & DVA, Positive and Negative Expected Exposures (PEE & NEE), and Collateral. AAD is used to calculate the bucketed risk for IR model and credit curves.

When run on a 2nd Generation Intel Xeon Scalable Processor, specifically on a system with two 28-core Intel Xeon Platinum processors (56 cores total) equipped with 192Gb of memory,[1] speed-ups of over a thousand-fold[2] were reached when using the full power of Intel AVX-512. As shown in Figure 1, with Intel AVX-512 plus memory optimizations due to caching, results in gains up to 1770x with 56 cores (112 threads) for xVA pricing. From this Figure 1, we can also see impressive 35X performance gain (compared to baseline) even on the single core. Figure 2 shows speed-up for xVA+Greeks calculations; gains for this application are up to 832x with 56 cores and are up to 23X gains on a single core.

One might ask "why did the performance increase drop for xVA+Greeks?" Honest benchmarking is the answer. The first graph (xVA pricing only) is a baseline vs. *Matlogica Vector Accelerator C++ Library* comparison. When computing Greeks with *Matlogica Parallel AAD-C C++ Library*, two things kick in: (1) use of AAD as a technique, and (2) the novel technology Matlogica has created to accelerate AAD runs. To highlight the latter, the baseline was increased to include AAD via the Adept library. Adept (Automatic Differentiation using Expression Templates) is an open source C++ software library. The increase in performance is especially remarkable since it uses Adept as a baseline. It would be tempting to use a bump-and-revalue as the baseline, in which case the performance gain would be particularly dramatic because of the advantage any AAD implementation would bring with it. Since bump-and-revalue method revalues original function for each input risk value, in this modest benchmark it could be expected to slow things down by factor of 140 if used instead of Adept—and the reported increase in performance would be much more if not for including Adept in the baseline.

## Integrating Matlogica Libraries to obtain impressive results

The *Matlogica Vector Accelerator Library* C++ uses operator overloading for integration with the user's code base. This allows the translation of even complex user valuations into binary code to harness the substantial benefits of Intel AVX-512.

Operator overloading used in conjunction with tracking variables dependent on function inputs provides an automatic way to optimize out all intermediate results that can be cached and hard coded into generated function for improved performance. On top of this, generated functions are by construction multithread safe by design, and do not require thread locking mutexes even when the original user program wasn't built with multithreading in mind.

The *Matlogica Parallel AAD-C C++ Library* is easy to integrate, without interfering with debuggability, into existing code because it requires minimal changes to utilize its operator overloading on numerical type (double). The operator overloading then easily extracts a valuation graph for additional processing. Without any additional programming, the AAD-C transforms this graph at runtime into efficient binary machine instructions isolated from the original program, which can even be easily sent for execution on a cloud farm. This allows use of Intel AVX-512 vectorization to its full capacity and produced code is ready for multicore and multithreaded usage (i.e., it is thread safe).

AAD-C consists of a collection of C++ overloaded functions—easy to integrate and designed to keep debugging straightforward. Automatic vectorization combined with a unique on-the-fly approach have yielded these outstanding performance results. For a small fraction of the original code execution time, you can get values of the base algorithm and all its derivatives.

AAD-C improves significantly on the prior techniques in multiple ways that can allow it to complete more analysis in less time. The observed thousand-fold performance gains stem from:

1. The translation to machine instructions once, instead of interpreting parts of a valuation graph repeatedly,
2. The ability to fully exploit Intel AVX-512.

Dmitri Goloubentsev, Head of Automatic Adjoint Differentiation at Matlogica LTD, shared that "With AAD-C programmers can develop their code in single threaded, predictable OO C++ and convert any heavy load block to flattened streamlined execution taking full advantage of Intel AVX-512 vectorization and threading. Even NUMA management is easy, because we have full control on memory blocks needed for execution. AAD-C helps to transform Object Oriented Design into Data-Oriented-Design".[6]

## Summary

Versatility of Intel Xeon Scalable Processors for a wide variety of Financial Services applications includes xVA. This fits well with the trends in many businesses to centralize xVA activities to better quantify and manage the costs to their derivatives business of counterparty credit, funding, margin, etc.

The benefits of Intel AVX-512 for applications with demanding compute needs are evident in these results. Many applications utilize the Intel® Math Kernel Library (Intel® MKL) for high performance from Intel AVX-512; the Matlogica libraries demonstrate strong results from their approach as well. Both libraries can be used, in the same application, as desired.

The *Matlogica Vector Accelerator C++ Library* and *Matlogica Parallel AAD-C5 C++ Library* results described here illustrate opportunities for thousand-fold performance improvements of reasonable baseline implementations. Best of all, these performance results are within reach for a wide variety of applications.

**Learn More**

Learn more about the AAD-C project.

Engage with Matlogica to see how they can offer quick proof-of-concept projects to gauge the possible benefits that the Matlogica Vector Accelerator Library can give to your specific applications.

Discover Intel HPC and AI technologies.

Request a demo.

[1] System Configuration: Two Intel Xeon Platinum 8280 processors (28 cores/processor) with a total of 56 cores (112 threads), 192GB memory, Matlogica AADC library (5/15 release) and Intel® C++ Compiler 19.1.1; xVA refers to X-Value Adjustment—an inclusive term for numerous different valuation adjustments used to assess pricing and risk associated with financial derivative instruments.

[2] Results published by Matlogica at www.matlogica.com

[3] AAD refers to Automatic Adjoint Differentiation (AAD)—a technique that greatly reduces the time to calculate sensitivities of derivatives prices to underlying factors, called Greeks.

[4] Greeks represent the sensitivity of the price of derivatives to a change in underlying parameters. The name "Greeks" stems from the fact that key sensitivities are known by their Greek letters including values such as spot price value (Δ delta), volatility value (*V vega*), and time to expiry value (Θ theta).

[5] The AAD-C name means Automatic Adjoint Differentiation (AAD) Compiler, where the term compiler refers to a compiler-like activity hidden from the user within the library and not a compiler in the traditional sense. This is part of the novel approach from Matlogica that yields such tremendous benefits.

[6] A nice introduction to Data-Oriented-Design can be found at https://youtu.be/rX0ItVEVjHc



0820/RJMJ/MB/PDF Please Recycle 44201-001US