Signaloid Computing Engine Amazon Machine Image

The Signaloid Computing Engine Amazon Machine Image (AMI) provides the Signaloid Computing Engine SDK virtualization layer and the associated utilities needed to compile and run C/C++ applications to take advantage of Signaloid's UxHw technology. By using the Signaloid Computing Engine AMI and the Signaloid UxHw® SDK, you can deploy to your existing AWS infrastructure accelerated execution of stochastic workloads such as Brownian motion path numerical solution of stochastic differential equations. You can deploy instances on-demand and maintain full control over your compute environment, without the need to use the Signaloid REST API for launching workloads. To achieve these benefits, you only need to replace your default AMI with the Signaloid Compute Engine AMI and recompile your applications using the Signaloid UxHw SDK.

Installed Software

The Signaloid Computing Engine AMIs come pre-configured with the software required to compile and run applications using the Signaloid UxHw SDK.

The default username for all AMIs is `ec2-user`.

In the home directory of `ec2-user`, under `Signaloid-Demo-TemplateBuildAssets`, you can find template assets for building C/C++ applications using the Signaloid UxHw SDK.

The table below summarizes the target architecture, operating system, and pre-installed software of the available Signaloid AMI options.

Target Architecture	Operating System	Pre-installed software
AMD/Intel x86_64	Amazon Linux 2023.10.20260105	• Signaloid UxHw SDK v5.0.5 • Signaloid `signaloid-python` v1.9 • Python 3.14
AWS Graviton	Red Hat Enterprise Linux 10 (HVM)	• Signaloid UxHw SDK v5.2.1 (arm64) • Signaloid `signaloid-python` v1.9 • Python 3.14

Important: The Signaloid Computing Engine AMI comes without a pre-installed license for the Signaloid UxHw SDK. Contact Signaloid sales at `sales@signaloid.com` to request a license file for your AWS account, then install the license file at `/etc/Signaloid-UxHw-SDK/license.dat` on any EC2 instances that you deploy.

Quick Start Guide

The following steps provide an example of how to build and run a C-language application using the Signaloid UxHw SDK. You can follow the same steps to build and run your own C/C++ applications and take advantage of Signaloid's UxHw technology.

Value at Risk (VaR) Calculation Using a Geometric Brownian Motion (GBM) Process

Value at Risk (VaR) is an important quantitative risk metric for financial and insurance institutions when judging the risk of loss for a potential investment. The following steps use an example from the "Mathematical Modeling And Computation In Finance: With Exercises And Python And Matlab Computer Codes" [1] by Oosterlee and Grzelak, that implements a numerical solution of the stochastic differential equation (SDE) for a geometric Brownian motion (GBM) process and uses the distribution of the instrument price at maturity to compute the VaR. Such processes are traditionally evaluated across a set of paths (typically millions) and over a number of time steps (typically the 252 stock market trading days in a year).

When compiled with Signaloid UxHw SDK, the kernels implementing the GBM SDE can replace the usual approach of sampling followed by evaluation of each path for these sample inputs, for millions such paths, replacing it with a direct computation on a representation of the probability distribution across paths, while executing a single "distributional" path. Thus, in a single computation of the Brownian motion process, the UxHw-enabled code kernel can compute the same type of output distribution that would take a Monte Carlo simulation up to millions of iterations (paths) to compute.

Building the GBM Application Using the Signaloid UxHw SDK

1. Clone the GBM application using:

```
git clone --recursive
https://github.com/signaloid/Signaloid-Demo-Finance-OosterleeGrzelakBookGeom
etricBrownianMotionExample.git
```

The following guide assumes that you have cloned the repository into `/home/ec2-user/`.

To build the application using the Signaloid UxHw SDK, you need the template build assets that are in `/home/ec2-user/Signaloid-Demo-TemplateBuildAssets/`.

2. Copy the `Makefile.pro` file from the template build assets into the cloned repository:

```
cp
/home/ec2-user/Signaloid-Demo-TemplateBuildAssets/template/coreClass/C0Pro/Makefile.pro
/home/ec2-user/Signaloid-Demo-Finance-OosterleeGrzelakBookGeometricBrownianMotionExample/src/
```

Directory `/home/ec2-user/Signaloid-Demo-Finance-OosterleeGrzelakBookGeometricBrownianMotionExample/src/` should now contain the application source code and:

- `Makefile.pro`: Makefile for the configuration of the Signaloid UxHw SDK.
- `config.mk`: Optional input file for controlling the build process, e.g., defining `SOURCES` or `CFLAGS`.

3. Navigate to `Signaloid-Demo-Finance-OosterleeGrzelakBookGeometricBrownianMotionExample/src/` and build the application using:

```
cd
Signaloid-Demo-Finance-OosterleeGrzelakBookGeometricBrownianMotionExample/src;

make -f Makefile.pro REPRESENTATION_TYPE=TTR REPRESENTATION_SIZE=64
CORRELATION_TRACKING=COMMON_ANCESTOR
```

The above command builds the application for the C0Pro-S+ core of the Signaloid Cloud Compute Engine (SCCE). By default, `Makefile.pro` builds a binary with the name `main`.

Inputs and Outputs of the GBM Application

The following are the key financial parameters of the GBM application:

Parameter	Flag	Description
Initial Value	`--initial-value`	Starting stock/portfolio value ($)
Mean Return	`--mean-return`	Expected annual return rate
Volatility	`--volatility`	Annual volatility (risk measure)

Maturity Time	`--maturity-time`	Time horizon in years, e.g., 1.5.
Quantile	`--quantile`	VaR confidence level
Frequency	`--frequency`	Time step granularity: 0=daily (252 steps/year), 1=monthly (12 steps/year), 2=yearly (1 step/year)

The top-level README of the GBM application provides a complete description of the inputs and outputs of the application.

The examples in this quick start guide only calculate the price at maturity output of the GBM application (enabled using the `-S 0` command-line option), and the value at risk (VaR) output (enabled using the `-S 3` command-line option).

Example 1: Stock Price Distribution After One Year

Scenario: You own a stock currently worth $100. Based on historical data, it has:

- An expected annual return of 5% (0.05)
- Annual volatility of 40% (0.4)

Calculate the price distribution after one year with daily time steps:

```
./main --initial-value 100 \
      --mean-return 0.05 \
      --volatility 0.4 \
      --maturity-time 1.0 \
      --frequency 0 \
      -S 0
```

Output:

```
Stock price at maturity:
97.04Ux040000000000000001405A20D2F1D57933000000404043BE35B6824CF2012A40BE295
1F81040454FE5177FA65401386FA7DC4C72004046AA63F8909D37015EE7A0B2FAD2404047E8B
A8E1BFF5C017A70B91DF72D804049114398FB9755018E6F1C227DED20404A27CAD789A38501A
72A6C922D67A0404B2F73A24696C501B7500640BE8FC0404C2A44982092F101C55E222F7079D
0404D19B33FFA4D4401CFA3DBA64EB470404E005783179F4601DE20B28AE68B20404EE3004C5
2381101F4310C454D89C0404FBF95579EB83101F23AC3336AE9A040504B668B51BADA0204D48
A5828E5204050B5A1B85C4273020B3717808FFB6040511E8EF6350A9B0218E01EF6A4D900405
186FF00FB2B340222BF1E7921FA604051EE3992E370240223350AB479A960405253DCEC1E917
1022786EC060F2B204052B9D0C4670BF502380AE89426E0A040531EEE721391E2022BC858758
8CCC0405382E4258F4067023467FAF1EFC7804053E7D6D68BCD13023F941E31BEF14040544D0
6CFD0A6C7023C8A6190755B804054B2FADB2109DD024AC595FB6ACE0040551B38796567B3025
55B5D7CE52CA0405583CFE509474A024A7510598CA1A04055EC5856463AEA024F79D3521AAA4
040565569A93F2AD00248BB0889B1DD404056BEFDD9F96DB9024B8D3079F92C8040572AD5171
40F730254260C1781E3A0405798D930E0804D02531439226FF340405808687B0F21D20252C66
DBB8814A0405879DB74D7C62F0252D790ED66A9804058ED788778551C0252700148C02C80405
963737364728702516903 1E78BDC04059DE1DFB26A20F02646167F436FB00405A5BDD5A9F157
0024FD6503E39D380405ADB04F5580833024F95E74531E1E0405B5D2A9334E28E0247B0F6DF6
ECB40405BE31318AB6FF1024ACBF395A67640405C6D85F2B813F70244FD86473C4860405CFCC
82793C47D0245B16C5FCBD8A0405D920718524E16024469AA3551EE80405E2BDB91F044E6023
4F0C567DAB200405ECA14485ADF02022F82C6EC8524E0405F6F7DDD11F3E9022F3348855FBD4
040600F5888790B3F0231B83CEA772FC040606B6A6EFBA66002247C8D6DF2A4A04060CC830C9
8A03602252A5002A80AC040613341654C27200218BEF44492AE4040619F35F9F4B25602099BA
542A20CA0406211EA6CA52EA10203A0693AA0148040628C4D938DC5A601F148548B401200406
310BAFDCC45ED01F3839BDEC12E604063A0A76FA26DB001DC8D54074D79A040643C1AF45D8BB
601CEE980A29CA7A04064E7DB219405A501C4A98E747BE2804065A9BA9E7DF0CF01BC18A1292
4CFB0406685F2D64E058601A51DC2BAB4F9C040677D957D8283270186EB99011BE0A040689BC
226FDA766017653D61D4661F04069F49B5C76587701611528C2ECAF80406BA17D0AF46A72013
F4D6B54146350406DD473491E186E01258C73C6C887C0
```

The output contains output distributional information encoded in the form of Ux strings. To visualize the output of Ux strings, use the script in "Plot Ux Strings Using signaloid-python".

Example 2: Value at Risk Calculation

Scenario: You have a $1,000,000 portfolio and want to understand downside risk:

- Current value: $1,000,000
- Expected return: 5%
- Volatility: 15%
- Risk quantile: 5% (95% confidence level)

Calculate the Value at Risk using:

```
./main --initial-value 1000000 \
      --mean-return 0.05 \
      --volatility 0.15 \
      --quantile 0.05 \
      --maturity-time 1.0 \
      --frequency 0 \
      -S 3
```

Output:

```
Value at Risk:
-185270.63Ux040000000000000000C1069DB50B1662A400000001C1069DB50B1662A4800000
0000000000
```

Interpretation: With 95% confidence, your losses will not exceed $185,270.63 over the next year.

Plot Ux Strings Using `signaloid-python`

You can use the pre-installed `signaloid-uxdata-toolkit` of the `signaloid-python` package to plot a Ux string. The command-line option`-o` saves the output plot in PNG format:

```
signaloid-uxdata-toolkit plot -o price-at-maturity.png
97.04Ux040000000000000001405A20D2F1D57933000000404043BE35B6824CF2012A40BE295
1F81040454FE5177FA65401386FA7DC4C72004046AA63F8909D37015EE7A0B2FAD2404047E8B
A8E1BFF5C017A70B91DF72D804049114398FB9755018E6F1C227DED20404A27CAD789A38501A
72A6C922D67A0404B2F73A24696C501B7500640BE8FC0404C2A44982092F101C55E222F7079D
0404D19B33FFA4D4401CFA3DBA64EB470404E005783179F4601DE20B28AE68B20404EE3004C5
2381101F4310C454D89C0404FBF95579EB83101F23AC3336AE9A040504B668B51BADA0204D48
A5828E5204050B5A1B85C4273020B3717808FFB6040511E8EF6350A9B0218E01EF6A4D900405
186FF00FB2B340222BF1E7921FA604051EE3992E370240223350AB479A960405253DCEC1E917
1022786EC060F2B204052B9D0C4670BF502380AE89426E0A040531EEE721391E2022BC858758
8CCC0405382E4258F4067023467FAF1EFC7804053E7D6D68BCD13023F941E31BEF14040544D0
6CFD0A6C7023C8A6190755B804054B2FADB2109DD024AC595FB6ACE0040551B38796567B3025
55B5D7CE52CA0405583CFE509474A024A7510598CA1A04055EC5856463AEA024F79D3521AAA4
040565569A93F2AD00248BB0889B1DD404056BEFDD9F96DB9024B8D3079F92C8040572AD5171
40F730254260C1781E3A0405798D930E0804D02531439226FF340405808687B0F21D20252C66
DBB8814A0405879DB74D7C62F0252D790ED66A9804058ED788778551C0252700148C02C80405
963737364728702516903 1E78BDC04059DE1DFB26A20F02646167F436FB00405A5BDD5A9F157
0024FD6503E39D380405ADB04F5580833024F95E74531E1E0405B5D2A9334E28E0247B0F6DF6
ECB40405BE31318AB6FF1024ACBF395A67640405C6D85F2B813F70244FD86473C4860405CFCC
82793C47D0245B16C5FCBD8A0405D920718524E16024469AA3551EE80405E2BDB91F044E6023
4F0C567DAB200405ECA14485ADF02022F82C6EC8524E0405F6F7DDD11F3E9022F3348855FBD4
040600F5888790B3F0231B83CEA772FC040606B6A6EFBA66002247C8D6DF2A4A04060CC830C9
8A03602252A5002A80AC040613341654C27200218BEF44492AE4040619F35F9F4B25602099BA
542A20CA0406211EA6CA52EA10203A0693AA0148040628C4D938DC5A601F148548B401200406
310BAFDCC45ED01F3839BDEC12E604063A0A76FA26DB001DC8D54074D79A040643C1AF45D8BB
601CEE980A29CA7A04064E7DB219405A501C4A98E747BE2804065A9BA9E7DF0CF01BC18A1292
4CFB0406685F2D64E058601A51DC2BAB4F9C040677D957D8283270186EB99011BE0A040689BC
226FDA766017653D61D4661F04069F49B5C76587701611528C2ECAF80406BA17D0AF46A72013
F4D6B54146350406DD473491E186E01258C73C6C887C0

Signaloid Ux-data Plotter

==================================================


Parsing Ux-data...
Successfully parsed Ux-data!
Particle value: 97.04
Mean: 104.51287504050269
Variance: 1683.7659850114514
Number of Dirac deltas: 64
Double Precision: True

Preparing plot data...
Generating plot...
✓ Plot saved to: price-at-maturity.png
```

Figure 1 shows the price at maturity calculated in Example 1.

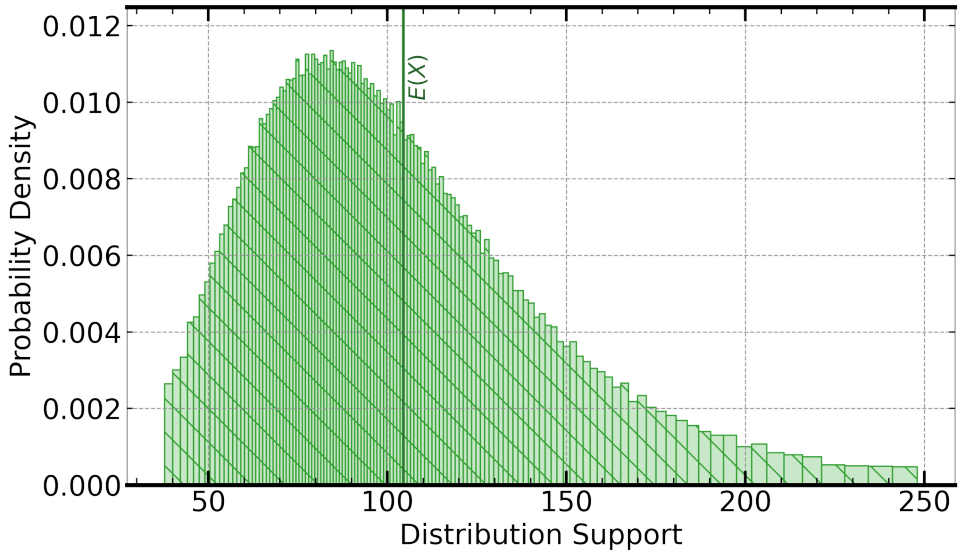

Figure 1: Price at maturity calculated in Example 1.

Performance

The geometric Brownian motion kernel, compiled with `REPRESENTATION_TYPE=TTR REPRESENTATION_SIZE=512 CORRELATION_TRACKING=CORRELATION_OFF` (same configuration as the C0Pro-XL core of the Signaloid Cloud Compute Engine) and running in this AMI deployed on an AWS r7iz instance, achieves performance 432x faster for the VaR workload, compared to an optimized C language Monte-Carlo-based implementation of the same kernel running on the same AWS r7iz instance without the Signaloid UxHw stack and achieving at least the same accuracy with 99% confidence. If requiring higher confidence in the accuracy of the Monte Carlo compared to UxHw, the speedups of UxHw are even greater. For more performance comparison details for the GBM kernel, see the Signaloid website.

Next Steps

- Contact us for pricing of the Signaloid AMI.
- Visit the Signaloid website for more information for developers and in-depth material on the Signaloid technology.
- Visit the Signaloid documentation page, to find out more about creating applications using the Signaloid UxHw API.

References

[1]: Oosterlee, C.W. and Grzelak, L.A. 2019. Mathematical Modeling And Computation In Finance: With Exercises And Python And Matlab Computer Codes. World Scientific Publishing Company. (Link to Python implementation).