MISYS
FINANCIAL SOFTWARE

Groovy and GPU : Enhancing pricing performance and quant productivity

Bram Leenhouwers
Senior Architect

Connect
Innovate
Expand

Misys at a glance

Who we are

A global leader in financial services software with

4,500+ employees

50+ countries.

“We are transforming the global financial services industry by making financial institutions more resilient, more efficient and more competitive.”

What we offer

The broadest and deepest portfolio of financial services software on the market. Misys solutions cover retail and corporate banking, lending, treasury, capital markets, investment management and enterprise-wide risk management

Who we serve

2K clients

46 of the world’s top 50 banks

12 of the top 20 asset managers

Our customers and their problem

Speed!

Our solution – FusionFabric

Misys FusionFabric

The high performance, scalable and resilient architecture

GPU Acceleration computing

- Massive parallelisation of computation-intensive tasks such as pricing
- Hardware agnostic
- Lower TCO vs traditional
- Grid computing

High performance computing:

- In-memory computing with native scalability
- No limit of computing nodes
- Lower TCO with commodity hardware

Fault tolerance and high availability:

Seamless fallover management with data and service redundancy for guaranteed business continuity

HTML

Multi device support:

Light and rich HTML5 user interface accessible from any device such as PC, smart phone and tablet

Real-time OLAP cube:

- Ultra-fast aggregation,
- Slice and dice on big data sets
- Real-time updates

Goal and challenges

Our goal

Run current and future pricing models on GPUs to dramatically speed up processing time for complex trading analytics and risk management calculations

Challenges

GPU code is complex to write

Requires specific skillset

Maintenance and extensibility is often difficult

Possible solutions

Software engineers	Quantitative analysts
Experienced in GPU progamming Less experienced in finance	Experienced in finance Less experienced GPU programming

Fusion parallel processor

Possible solutions

Each team gains experience in both fields ?

Migrate legacy code ?

Create an abstraction layer

Parallel Processing framwork concept

Software engineers

Build a scripting engine

Quantitative analysts

Write pricing models
Write payoffs

Translate a groovy script to OpenCL / CUDA / java bytecode

Easy to write and maintain

Runs fast

Parallelization

Easy to write and maintain

Runs fast

Where does the workload come from ?

- Price a whole portfolio of deals
- Price a deal using multiple market data (Montecarlo)
- Price a deal at several dates (PFE)

We separate the logic of the script from its parallelization axis

Framework overview

Quantitative analyst perspective

Script example

Script

```
// Retrieve rate
def rate = data0D(“INTEREST_RATE”, [currency1], fixingDate)

// Compute daycount fraction
def daycount = dayCountFraction(effectiveDate, maturityDate, currency1)

// Compute discount factor
def factor = discountFactor(calculationDate(), maturityDate, basis)

// Compute the price
def price = notional * daycount * (rate - fixedRate) * factor

// return price
return price
```

Deal

```
{
    "currency1": "USD",
    "currency2": “EUR",
    "fixingDate": "2015-03-17",
    "effectiveDate": "2015-03-01",
    "maturityDate": "2015-03-01",
    "basis": "ACTUAL_360",
    "notional": 1000000
}
```

Deal axis

Script example

Script

```
// Retrieve rate
def rate = data0D("INTEREST_RATE", [currency1], fixingDate)

// Compute daycount fraction
def daycount = dayCountFraction(effectiveDate, maturityDate, currency1)

// Compute discount factor
def factor = discountFactor(calculationDate(), maturityDate, basis)

// Compute the price
def price = notional * daycount * (rate - fixedRate) * factor

// return price
return price
```

Deal

```
{
    "currency1": "USD",
    "currency2": "EUR",
    "fixingDate": "2015-03-17",
    "effectiveDate": "2015-03-01",
    "maturityDate": "2015-03-01",
    "basis": "ACTUAL_360",
    "notional": 1000000
}
```

Deal axis **Market data axis**

Script example

Script

```
// Retrieve rate
def rate = data0D(“INTEREST_RATE”, [currency1], fixingDate)

// Compute daycount fraction
def daycount = dayCountFraction(effectiveDate, maturityDate, currency1)

// Compute discount factor
def factor = discountFactor(calculationDate(), maturityDate, basis)

// Compute the price
def price = notional * daycount * (rate - fixedRate) * factor

// return price
return price
```

Deal


```
{
    "currency1": "USD",
    "currency2": “EUR",
    "fixingDate": "2015-03-17",
    "effectiveDate": "2015-03-01",
    "maturityDate": "2015-03-01",
    "basis": "ACTUAL_360",
    "notional": 1000000
}
```


Deal axis **Market data axis** **Date axis**

Script example

Script

```
// Retrieve rate
def rate = data0D(“INTEREST_RATE”, [currency1], fixingDate)

// Compute daycount fraction
def daycount = dayCountFraction(effectiveDate, maturityDate, currency1)

// Compute discount factor
def factor = discountFactor(calculationDate(), maturityDate, basis)

// Compute the price
def price = notional * daycount * (rate - fixedRate) * factor

// return price
return price
```

Deal


```
{
    "currency1": "USD",
    "currency2": “EUR",
    "fixingDate": "2015-03-17",
    "effectiveDate": "2015-03-01",
    "maturityDate": "2015-03-01",
    "basis": "ACTUAL_360",
    "notional": 1000000
}
```


Deal axis **Market data axis** **Date axis** **User defined functions**

Language syntax highlights

Static type inference

No explicit typing, everything is detected and optimized at compile time

Natively supported types : Double, String, Date, Boolean, Array, Matrix, Cubes…

Standard flow operators

for, while, if, else, switch/case, break, continue…

Custom functions

Function declaration with typeless parameters

Custom structures

Class-like structure definitions

Function pointers

Through seamless static templating

Standard library

A set of optimized standard functions and algorithms provided by default

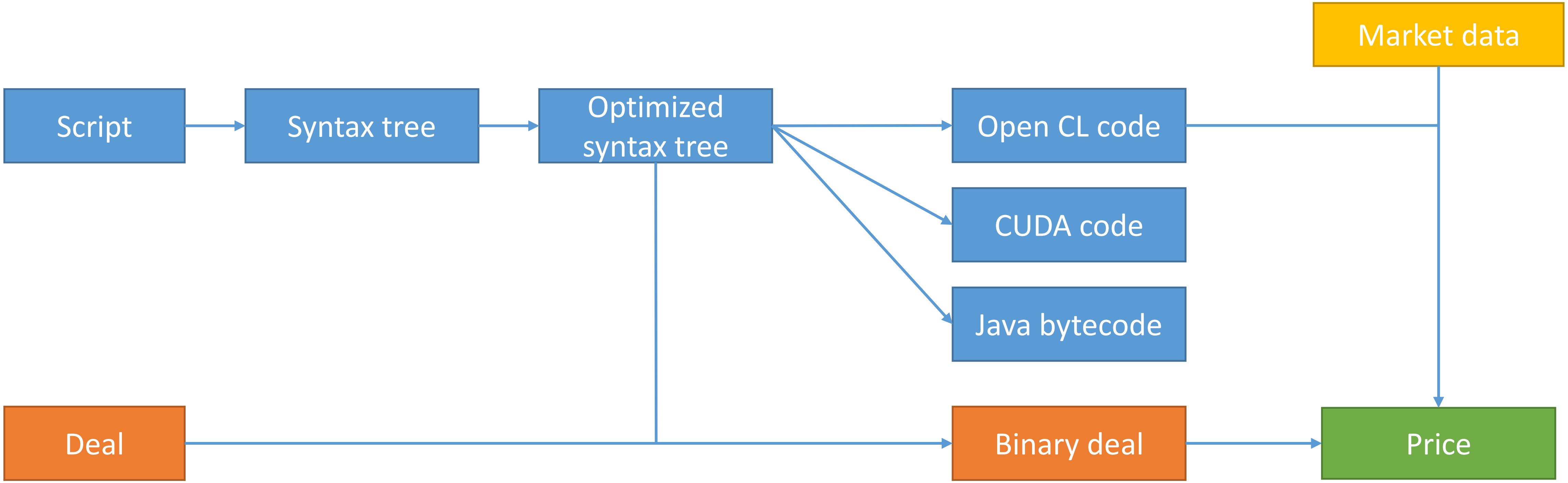
Code demonstration

Connect
Innovate
Expand

Framework overview

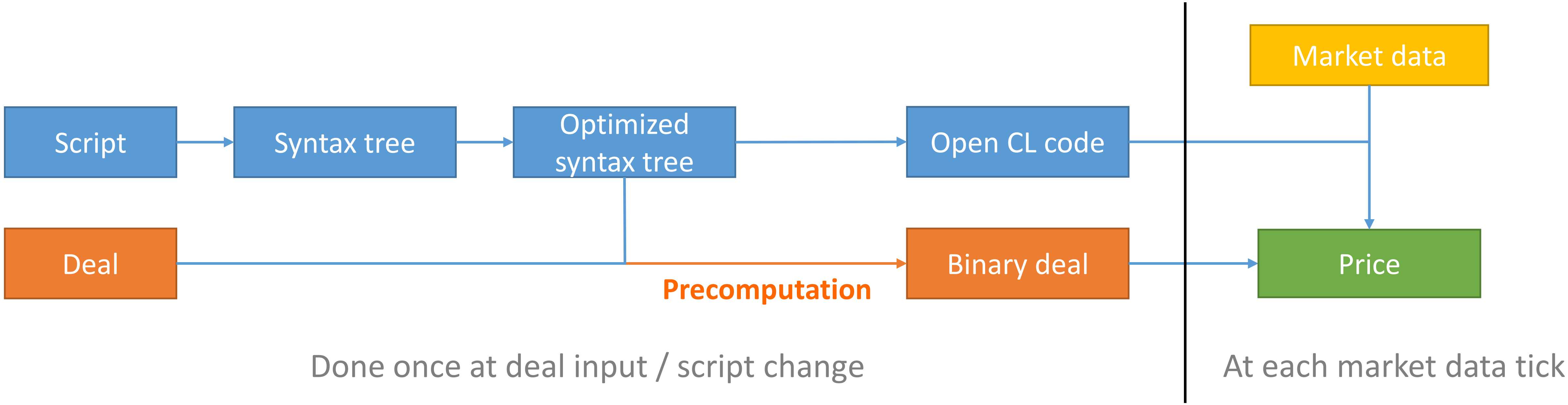
Software engineer perspective

Framework overview



Precomputation

If you can do it only once, don't do it every time !



Precalculate everything that is independent from market data and computation date

Precomputation

Compute invariants only once

Script

```
// Retrieve rate
def rate = data0D("INTEREST_RATE", [currency], fixingDate)

// Compute daycount fraction
def daycount = dayCountFraction(effectiveDate, maturityDate, currency)

// Compute discount factor
def factor = discountFactor(calculationDate(), maturityDate, basis)

// Compute the price
def price = notional * daycount * (rate - fixedRate) * factor

// return price
return price
```

Precomputation

Not depending on any market data
Not depending on calculation date
daycount is precomputed

notional comes from the deal
daycount is precomputed
notional * daycount is precomputed

Migrating from legacy

Existing C++ / Java code cannot be magically parallelized…
But we can call it during the precomputation phase !

Limitation: the inputs must be independent from market data
(The market data will be different for each scenario)

Migrating from legacy (2)

Script

```
// Retrieve rate, each execution can have a different value
def rate = data0D("INTEREST_RATE", [currency], fixingDate)

// Compute daycount fraction
def daycount = dayCountFraction(effectiveDate, maturityDate, currency)

// Compute discount factor
def factor = discountFactor(calculationDate(), maturityDate, basis)

// Compute the price
def price = notional * daycount * (rate - fixedRate) * factor

// return price
return price
```

Precomputation

Not depending on any market data
Not depending on calculation date
dayCountFraction can be in **Java**

Performances

The framework takes care of:

- Data alignment / Coalescence
- Memory copy
- Optimized occupancy
- Multi GPU
- Float or double precision
- Separation of CPU / GPU execution through precomputation
- …

Optimizing the engine optimizes all existing scripts

Performance demonstration

Connect
Innovate
Expand

Summary

Scripting language
Easy to code, read and maintain scripts

Same script, multiple use cases
Can be used for 3D simulation, Monte carlo pricing, PFE, CVA…

Progressive migration
Through the ability to call legacy functions in java or any other native language

Performance
Float or double precision and multiple GPU support

Tools available
Debugger, non regression framework

Bram Leenhouwers
Senior Architect
bram.leenhouwers@misys.com

misys.com

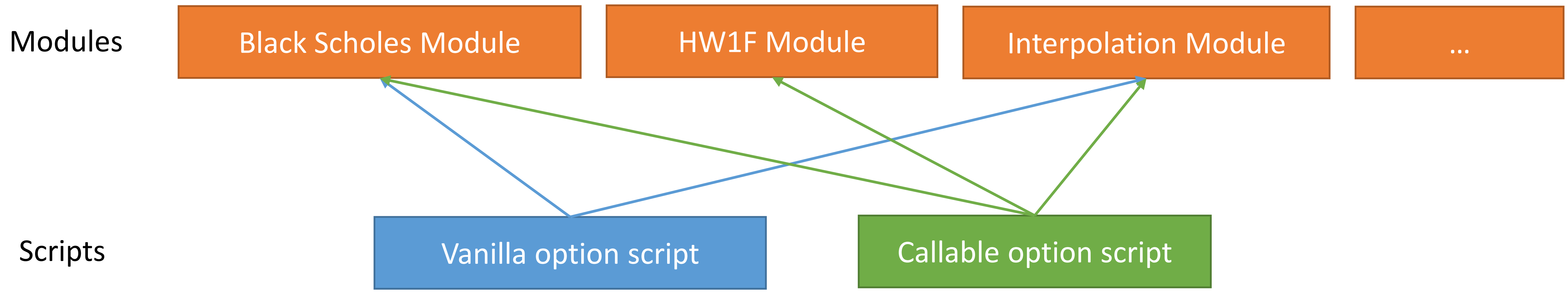
Please consider the environment before printing this PowerPoint.

@MisysFS

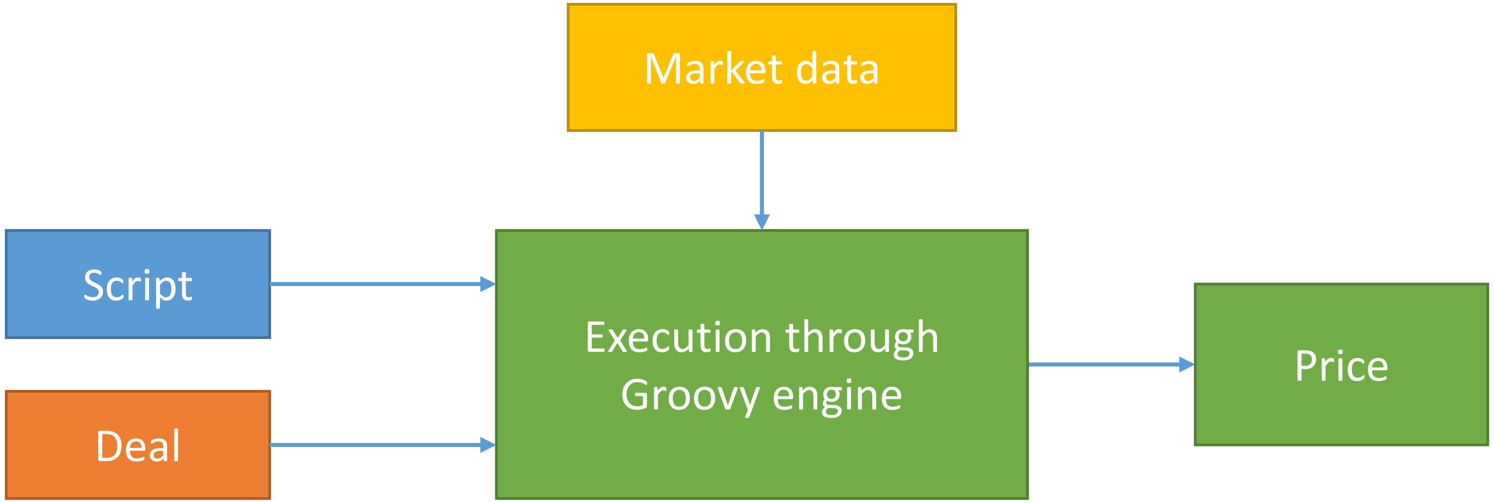
Misys @ LinkedIn

MisysVideoChannel

Code sharing and re-usability



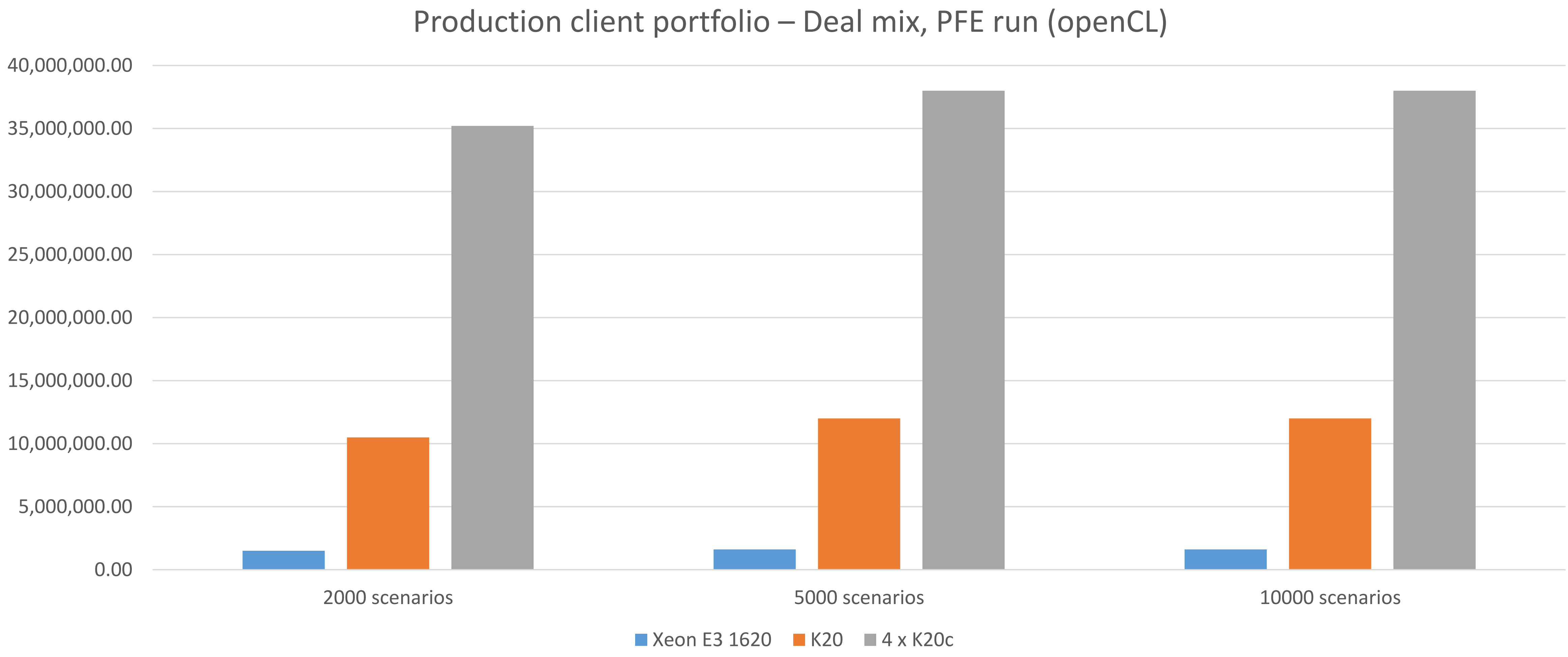
Debugging



Debug mode uses one deal and one market data setup

Allows breakpoints and variable introspection

Benchmark


Production client portfolio – Deal mix, PFE run (openCL)
40,000,000.00
35,000,000.00
30,000,000.00
25,000,000.00
20,000,000.00
15,000,000.00
10,000,000.00
5,000,000.00
0.00
2000 scenarios
5000 scenarios
10000 scenarios
Xeon E3 1620
K20
4 x K20c

Parallel Processing Framework pillars

Unified
Single pricing platform within Misys

Versatility
Must handle all pricing models, not only vanilla products

Performance
GPU based for computation intensive tasks

Adaptability
Migration path from existing pricing models

Transparency
Easily develop, expose and maintain pricing models

Portability
Be hardware / OS / technology agnostic