NIMo Reference Server

These are plans for a generic open source –o2 gcc dev server with a fully AVX2 optimized option for full balance sheet simulation. This generic simulation code will be a reference for a Consortium of banks and financial service firms to promote standards development, competitive performance, and computational stability in CCAR balance sheet simulation. Kintex will maintain AWS NIMo Reference Servers for Firms to check their internal balance sheet simulation results. Kinetix will also provide code and expertise to the US Federal Reserve to assist in aspects of the oversight of the financial system with up-to-date competitive balance sheet simulation tools. The Consortium will regularly solicit improvements to the basic design, technology standards, and quantitative analytic implementation of the NIMo Reference Server from Consortium members, academics, and practitioners by maintaining an Open Source policy. The Balance Sheet data, the operations models, the security models, and the detailed code optimizations required to make the balance sheet simulation's performance competitive and relevant to a particular Firm's capital planning and regulatory reporting, obviously remain wholly controlled by the Firm.

The reason Firms work with the Consortium and Kinetix is raw simulation performance applied to their balance sheet. In 2016 the NIMo Reference Server runs a 5-year full CCAR regression-fit simulation of a multi-trillion dollar balance sheet on a single commodity x86 processor in several seconds. If the Reference code is optimized it can run 10 to 20 times faster on a single microprocessor core. If a bank needs that kind of world-class competitive performance they run it with their internal HPC groups or work with the Consortium. The Consortium uses commodity microprocessors, compilers, and development tools so the Firm's simulations can be maintained and debugged by conventional means. Given a modest size multiprocessor running the optimized simulation code the entire annual capital allocation plan for a large bank can be automated and optimized with a dynamic feedback system.

The Reference NIMo Server provides static worst-case and stochastic expected case simulation, the later with an LMM stochastic market model (see Sandberg, *Finding NIMo*). We also will run the computation to optimize the capital allocation plan over a selected simulation horizon. The server provides all the security and simulation model selections preconfigured into an optimized simulation that the user selects. User provides the current and forward market data, the accrual portfolio, and the initial capital allocation plan.

This is a generic C++ code implementation and benchmarking of a full balance sheet simulation on a Haswell microprocessor (read old commodity x86 microprocessor). Other than the architecture of the server software, there is nothing special about

this completely unoptimized reference code. We expend some effort to keep the floating point cycles together and maintain a software architecture that will optimize well through a commodity memory hierarchy. Note we are using gcc here rather than a state of the art optimizing compiler. The code is under development and is available in GIT see https://github.com/jsandber/finding_nimo.

Given how long CCAR worst case simulation development has taken across the Street, NIM capital plan optimization from the accrual portfolio position level is not automated nor optimized at any top bank. There may be some optimization at a higher aggregated level across business lines for the accrual portfolio. Typically, you do not expect a manual and unoptimized process to even reach 90% of optimal. If NIMo moves the NIM from 300bps to 310bps the revenue gain to a large bank approaches $2Bn annually, that's for a third of one percent improvement on the existing manual capital plan. The existing manual and unoptimized capital plan would need to be 99.6 % of optimal both operationally and in the expected market for that improvement to not be on the table at each bank reporting CCAR. That is completely implausible at any of the large CCAR banks although the data and now the computational capacity are available.

If the accrual portfolio positions get reported daily (or even weekly) there is typically enough computational power available in the existing stock of commodity microprocessors to run a fully automated daily global NIM optimization. Looking at the problem, we observe both an operational and market level opportunity to optimize. Moreover, it is the operational side for many banks that seems to make this very likely to be significantly profitable. Note that taking on positions in deposits, cards, or mortgages are not typically accomplished through two sided markets. There is no exchange to trade in and out of deposits or credit cards. There is an operational element to entering these positions. If the capital plan requires the bank to enter into 1Bn USD of credit cards in California there is a stochastic process depending on branch operations in California that will determine how much and when the bank will have the target new investment card position inventory. There is no exchange providing liquidity for a spread that assures the bank enters into the desired cards position. The idea in NIMo is to model the operational execution ability of the branches charged with implementing parts of the capital plan. Optimize the allocation of capital according to the stochastic model of the branch execution in addition to the customer and market (rates and credit) dynamics.

This Balance Sheet simulation Reference Server design summary covers the problem size and performance expectations in the Background section. The next section, Server Data Model, covers the data layout and class structure for the reference server. The section IO Functions lists the initialization and reporting interface to the server classes. The Compute Functions section lists the external API entry points and the internal functions for product model specification and stochastic market model calibration. Finally the Test Usage section outlines the development stages for this balance sheet simulation reference server from the hold flat model and static balance sheet simulation to fully optimized capital allocation.

Background

Global Banks are in the process of developing their own Reference Severs for CCAR balance sheet simulation. They are facing several harder problems than simply optimizing the balance sheet simulation code. They have to track down the global accrual portfolio, track the right level of security aggregation, provide sensible baseline global balance and rate models, and report to federal regulators as well as senior management. If their balance sheet simulation performance is a couple of orders of magnitude off competitive performance that is not such a big deal at this point of the development cycle. We expect the reference server described in this document, to stay within an order of magnitude of competitive performance on a contemporary commodity microprocessor. The reference code is a little easier to manage compared to the fully optimized version. Once the reference code is stabilized we will optimize it for a particular target multiprocessor.

How big can the NIMo problem get? What are the maximum size estimates?

We assume the balance sheet is a series of quantities per account (balance, rate of return, and default) simulated monthly out 5 years. The number of accounts is under half a million all in. Assume 80% of the face amount of the accrual portfolio is typically in about 10 to 20 thousand of those hierarchical accounts. We assume new investments at any given point of time is around 10K parameterized positions. The total number of securities or equivalently securities with distinct balance, rate, or default models is under 100K (this is an important estimate because this determines how fast we can simulate, the rest is just shuffling data around). We assume the securities are deposits, fed funds, cash, cards, commercial loans, some UST/Sov and MBS held in AFS/HTM, and some long-term debt. Cards and deposits are probably aggregated from something like 100mm to 500mm individual contracts. So for trillion dollar or larger balance sheets we expect under 100mm positions, probably closer to 5mm or 10mm. Computationally we are fine with 500mm or even 1 billion contract accrual portfolios if the need arises for a client.

The Net Interest Margin optimization problem has two parts:

1. an expected case Monte Carlo simulation of the Runoff portfolio balance, rate, and default models per security model and
2. an LP/NLP optimization of the identity and timing of entering into new investments over the simulation horizon.

As new investments roll into subsequent simulation periods they become part of the runoff portfolio and require entry into the Monte Carlo balance sheet simulation. The asymptotic runtime complexity for the LP/NLP step is cubic in the best case so that was the initial worry in our research. As it turns out, the LP/NLP step does not heavily depend on the long-only runoff accrual portfolio (a slight simplification since there are AFS positions that can be sold off and the accrual portfolio does hold interest rate and FX hedges). The operating assumption is that the size of the LP/NLP problem is closer to ~10K new positions than to the ~100mm runoff

contracts. If that assumption holds in practice then the LP/NLP can run on a single relatively small multiprocessor in an hour or so.

The Monte Carlo simulation is going to consume some hardware if we go to daily simulation periods in year 1. But the simulation code is very straight-forward to optimize once the spec. settles down if you understand how contemporary commodity microprocessors issue and execute FP instructions. That is why we delayed the server implementation until now. The trick with the code is to get competitive performance on a single core before branching to parallel processes. The current estimate is a trillion dollar balance sheet NIMo will require several hours on upwards of 1000 cores. If you do not get the single core optimization competitive you are going to need something like a million cores for several hours or you have to change the parameters of the problem (check your current Reference server balance sheet simulation performance, your mileage may vary). The optimization of the Monte Carlo simulation code is what makes NIMo tractable for a trillion dollar balance sheet in the sort term

To get a sense of performance scale, we implemented the hold flat balance sheet simulation across balance sheets ranging from 1K accounts to 512K accounts and timed the simulation execution (see Figure 1). At a half a million accounts the hold flat balance sheet simulation requires about 0.2 seconds on a single iMac core.

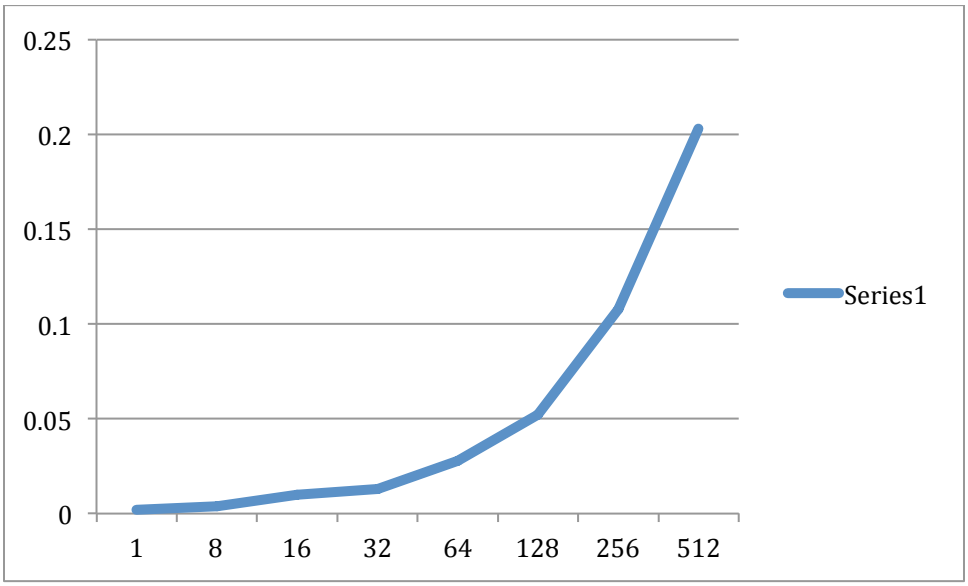

Figure 1: Balance Sheet Hold Flat Simulation seconds vs. thousands of accounts

The reference code is straightforward and more or less unoptimized. The code runs on a single thread of a single core of a 4 core Intel Core I7 clocked @ 4GHz with 16 GB of DDR3 memory. The balance sheet arrays are dynamically allocated off the heap.

```
int BalanceSheet::simulate(long max_acct, int max_periods, float
* balance, float * rate) {

    for (int i = 0; i < max_acct; i++ )
        for (int j = 0; j < max_periods; j++ ) {
            *(this->balance + i*this->num_cols  + j)= balance[i];
        }
    for (int i = 0; i < max_acct; i++ )
        for (int j = 0; j < max_periods; j++ ) {
            *(this->rate + i*this->num_cols + j)= rate[i];
        }
    return(0);
}
```

Figure 2: Balance Sheet hold flat C++ Reference Code

Look at Z Smith's bandwidth benchmarks for L1 caches to get a sense of where reasonable competitive performance is currently. The code I ran fills two arrays rate and balance with the initial balance sheet levels out 60 months, holding all levels flat. The arrays at the largest are 512K (accounts) * 64 (monthly simulation periods) * 4 bytes (float). So 256MB per array or 512 MB all in written in 0.2 seconds. Call it 2.5 GB/second. The high-end 4GHz box in Z Smith's table shows 64GB/sec for L1 read and write bandwidth – so the Reference Server unoptimized performance numbers make sense. There is about a factor of 25 left to optimize away from the reference implementation on a single core of my old iMac.

OS	Transfer size	PC Make/model	CPU	CPU speed	Front-side bus speed	L1 read MB/sec	L1 write MB/sec	L2 read MB/sec	L2 write MB/sec
Intel Linux 64	128 bits		Intel Core i7-930	Overclock 4.27 GHz	2000 MHz	64900	65100	43200	39900
Mac OS/X Snow Leopard	128 bits	Macbook Pro 15 2010	Intel Core i5-520M	2.4 GHz	1066 MHz	44500	44500	29600	27300
Intel Linux 64	128 bits	Lenovo Thinkpad T510	Intel Core i5-540M	2.53 GHz	1066 MHz	42000	42000	28500	26500
Mac OS/X Snow Leopard	128 bits	Macbook Pro MC374LL/A	Intel Core 2 Duo P8600	2.4 GHz	1066 MHz	36500	34500	17000	14300
Intel Linux 64	128 bits	Thinkpad Edge 15	Intel Core i3-330M	2.13 GHz	1066 MHz	32110	32070	21380	19730
Intel Linux 64	128 bits	Toshiba L505	Intel T4300	2.1 GHz	800 MHz	31930	30190	15000	12500
Intel Linux 64	128 bits	Toshiba A135	Intel Core 2 Duo T5200	1.6 GHz	533 MHz	24250	18970	9619	7237
Intel Linux 32	32 bits	Lenovo 3000 N200	Celeron 550	2.0 GHz	533 MHz	7489	7125	6533	5007
Intel Linux 32	32 bits	Toshiba A205	Pentium Dual T2390	1.86 GHz	533 MHz	7098	6734	7095	5675
Intel Linux 32	32 bits	Acer 5810TZ-4761	Intel SU4100	1.3 GHz	800 MHz	4937	4682	4160.	3013
Intel Linux 32	32 bits	Dell XPS T700r	Pentium III	700 MHz	100 MHz	2629	2284	2607	1630.
ARM Linux 32	32 bits	Sheevaplug	Marvell Kirkwood ARM	1.2 GHz		3418.	529.0	469.6	859.1
Windows Mobile	32 bits	HTC Jade 100	Marvell ARM	624 MHz		2165.	483.7		
Intel Linux 32	32 bits	IBM Thinkpad 560E	Pentium MMX	150 MHz	Up to 66 MHz	500.7	75.49	520.6	74.81

Figure 3: Z Smith Bandwidth Benchmarks

The following references cover aspects of optimizing codes implementing memset()and memcpy() functionality.

http://www.xs-labs.com/en/blog/2013/08/06/optimising-memset/

https://software.intel.com/en-us/articles/memcpy-memset-optimization-and-control

http://zsmith.co/bandwidth.html

There are a several things to keep in mind as we ramp up this Reference Server balance sheet simulator. 1. The FMA FP execution units are totally idle in this benchmark, it is easy to slip in code for valuing regression fits without changing he performance all that much; 2. The balance sheet representation may be too simple here and folks may want double rather than float (all in about a 6x performance effect – 512K accounts in 1.2 seconds unoptimized hold flat); 3. For efficiency reasons the balance sheet simulation should really be a distinct product simulation that is then distributed to the positions in the balance sheet; 4. This is all running on one thread on one core of a cheap old commodity microprocessor.

Server Data Model

nimo_configuration: Overall configuration and status data for nimo server.

- Select code optimization level
- Select static scenario or stochastic scenario model
- Select post simulation optimization level
- Balance sheet simulation configuration
- Monte Carlo simulation configuration
- IO Status reporting
- Market scope limits
- Log level specification
- Server provided selections with some user parameterization.

nimo_model_master: Models for aggregated NIM reporting or optimization of NIM.

- Lists post simulation aggregation of LP/NLP optimization level
- Lists new investments for NIM simulation
- Models are statically compiled into the server code.
- Dynamic Programming models for segregated capital plans
- Server provided selections

capital_plan_model: Capital allocation plans and implementation history.

- Lists capital allocation models
- Log of capital allocation implementation decisions
- Server provided with user parameterization.

accrual_portfolio: The Accrual Portfolio security positions per account on as of date.

- Lists positions in securities as seen in security master
- Maps positions to accounts (enumerating the balance sheet)
- Once security master simulation is complete the accrual portfolio shows how to scale, distribute and aggregate the simulated quantities.
- User supplied Portfolio and account mapping.

security_master: Forward looking non-market and macro data needed for security model evaluation and forward simulation wrt as of date. Indexed by the current market, a static scenario, or a stochastic scenario. Segregated into rolloff and new investment securities.

- Lists securities, indicatives, and aggregation status
- Assets: Loan, Deposit, Fed Fund, Treasury Bond (Investments)
- Liabilities: Deposit, Fed Fund, and LTD
- Maps securities to historical levels, balance, rate, and default models in security_model_master.
- Rolloff and new Investment securities.
- Stores balances, rates, and realized default historical and projected levels as listed in security model master.
- Projections on 5Y monthly simulation horizon
- Simulation target is security master
- Server provided selections.

security_model_master: Security level balance sheet simulation models and historical calibration data wrt as of date.

- Target balance, rate, and default models for securities in security_master.
- Historical data for model calibration
- Models are statically compiled into the server code to initialize the security master.
- The security model count determines the number of securities all securities with the same model are the same security.
- Simulate 10 models initially for each of the 10 accounts (small)
- Simulate par amount in security master and scale to actual face amount in the balance sheet. Generate TP in the balance sheet.

current_market: Market and macro levels for user selected as_of date.

- Market and macroeconomic levels required for balance, rate, and default model evaluation.
- Restricted to USD initially
- Restrict all model regression variables to 10Y swap and CPI initially.

- Required historical levels and projections out 5Y at 1 month intervals.
- Server provided variables – user supplied market and macro levels.

scenarios: Forward static valuation of std. market scenarios to simulation horizon wrt as of date. Expected case MC simulation models.

- Current and forward perturbations to current market and macro levels
- Monte Carlo Simulation market models

balance_sheet: Current and forward balance sheet levels agg'd. at the account level corresponding to one of current market, a static scenario, or a stochastic scenario.

- Account level indexed aggregation of balance, rate, and default
- Aggregated Balance subdivided into Retained to the end of simulation period, Rolloff by the end of the simulation period, and Acquired by the end of the simulation period
- Balances, Rates, and Defaults can be positive or negative
- Balance sheet is purely USD to start – ext. to multi currency later.
- Balance sheet simulations come in two sizes big: (100K accounts) and small (10 accounts)
- All balance sheet simulation horizons are 5Y monthly (60 double terms)
- Balance sheet is simply a target for scaling and sub-aggregation from the security master. You don't actually simulate the Balance Sheet. You do, however, optimize the capital plan based on the constraints induced by the placement of securities in the balance sheet (e.g., probably will not fund across tax jurisdictions).

IO Functions

User supplies all the required data for each simulation and no user state is persisted between simulations. Users provide the accrual portfolio, the current market, the capital plan and implicitly the accounts. The server simply assumes the accounts are a set in integers up to a user specified maximum value. The user is responsible for any desired account aggregation external to the server. The server just simulates the balance sheet. The reason we do not eliminate the accrual portfolio as an input in favor of a list of distinct securities is that the capital plan optimizer is likely to need to use the geographic and subsidiary structure to dynamically program NIMo. So, we keep the full accrual portfolio as an input to the reference server.

The user can designate preconfigured specific securities for the simulation including accrual inventory and new investments.

1. **get/set nimo configuration** – server configuration
2. **get/set accrual portfolio** – accrual portfolio

3. **get/set security master** – identify inventory and new investments
4. **get/set balance sheet** – 2d array of account X simulation time period
5. **get/set capital plan** – schedule of potential new investments and stochastic branch operations models.
6. **get/set current market** – market and macro variable levels required to cook and evaluate balance, rate, and default models
7. **get/set perturbed market** – market and macro variable levels required to cook and evaluate balance, rate, and default models with specified perturbations (e.g., parallel shifts or CCAR scenarios).

Compute Functions

The compute functions fall into two categories: externally facing user invoked functions and internal functions supporting the offered services.

External API Functions

1. **initialialize nimo** – server configuration
2. **cook_market_data** – interpolate market levels from benchmarks for required levels, termstructures and surfaces.
3. **initialize_balance_sheet** – compute the initial balances, return rates, realized defaults for the first time period of the simulation.
4. **simulate_balance_sheet** - Simulate the evolution of the balance sheet over a simulated passage of time using either static or stochastic market/macro scenarios.
5. **optimize_capital_allocation** – Find the optimal capital allocation plan for the expected operations/market/macro environment.

Internal Functions

Server needs to have the capacity to cook or interpolate all the required market and macro economic data to move the simulation forward. Additionally, the server must provide all the security balance, rate of return, and default models that the user may select. The server must implement all the stochastic market models driving the MC simulation. The server must implement all the LP/NLP optimization algorithms for capital allocation optimization.

1. **Regression Fit model** - Regression fit security models to specific market/macro variables
2. **Stochastic Market model** – code to implement LMM MC simulation, MC optimization.
3. **Capital Allocation Optimization model** – code to implement LP/NLP optimization of capital allocation
4. **System Specific Benchmarking** – performance variations between Haswell, Broadwell, and Skylake servers – tune code to specific commodity server architecture.

5. **FP Timing analysis –** look for algorithmic optimizations based on the portfolio/security model/ market data and simulation horizon.
6. **Portfolio size scaling analysis** – scale up to billion security accrual portfolios
7. **Security Model performance scaling** – scale up quantitative security models from regression fit to control the approximation error.

Test Usage

These are the sequences of user API function calls to run specific simulations on the server.

The function calls in bold typeface are the only times optimization makes a difference in scaling up the computation to take a billion security accrual portfolio. We may choose to collapse these calls into single a single piece of optimized code. Note that means we always take the time to pass the accrual portfolio and the market over a wire as the rate limiting step to any performance optimization. Maybe we relax this later?

- Select the as of date
- Set the simulation horizon to 60 monthly periods from the as of date.
- Get 60 forward monthly 10Y swap rates and CPI levels for user selected market/scenario.
- Get max sec_id – determine the number of different securities in the accrual portfolio – allocate memory.
- Get max acct_id – determine the size of the balance sheet.
- Get the mapping of securities to accounts in the accrual portfolio.
- Set SecurityModelMaster(scenario_id, as-of_date) so we fit regressions to the historical product data to obtain regression coefficients.
- Simulate securities id'd in step 2 forward 60 months from the as of date according to the scenario/market selected in step 1.
- Assign abd aggregate in the balance sheet per the accrual portfolio mapping selected in step 4.
- Return the simulated Balance sheet.

1. Hold flat USD balance sheet simulation 5Y monthly horizon.
    a. initialize nimo
    b. set current market
    c. set accrual portfolio
    d. set nimo configuration – hold flat
    e. initialize balance sheet
    f. **simulate balance sheet**

   g. get balance sheet
   h. stat nimo
2. Full model balance sheet simulation w. 5Y horizon at base market
   a. initialize nimo
   b. set current market
   c. set accrual portfolio
   d. set nimo configuration – full simulation
   e. cook market data – for balance, rate, and default functions (not discounting)
   f. initialize balance sheet
   g. simulate balance sheet
   h. get balance sheet
3. Full model balance sheet simulation base, +- 100, +- 200 at 5Y horizon
   a. initialize nimo
   b. set current market
   c. set accrual portfolio
   d. initialize Base +- 100 and 200 scenarios
   e. set nimo configuration – full simulation base, +- 100 and 200
   f. cook market data – for balance, rate, and default (not discounting)
   g. initialize balance sheet
   h. simulate balance sheet
   i. get balance sheet
4. Expected Monte Carlo model balance sheet simulation from base at 1Y horizon.
   a. initialize nimo
   b. set current market
   c. set accrual portfolio
   d. initialize stochastic scenario
   e. set nimo configuration 1Y LMM
   f. cook market data
   g. initialize balance sheet
   h. simulate balance sheet
   i. get balance sheet

5. Expected Monte Carlo model balance sheet simulation@ 1Y horizon w. NLP optimized capital allocation plan.
   a. initialize nimo
   b. set current market
   c. set accrual portfolio
   d. initialize stochastic scenario
   e. initialize capital plan model
   f. set nimo configuration 1Y LMM
   g. cook market data
   h. initialize balance sheet
   i. simulate balance sheet

j. optimize capital allocation plan
k. get balance sheet
l. get capital plan