NIMo Reference Server

These are plans for a generic open source Net Interest Margin optimization (NIMo) server that will support full static and stochastic balance sheet simulation for liquidity management and capital structure optimization (see Figure 1). This generic simulation code will be a reference for a Consortium of banks and financial service firms to promote technical standards development, competitive floating-point performance, and computational stability in balance sheet simulation/optimization.

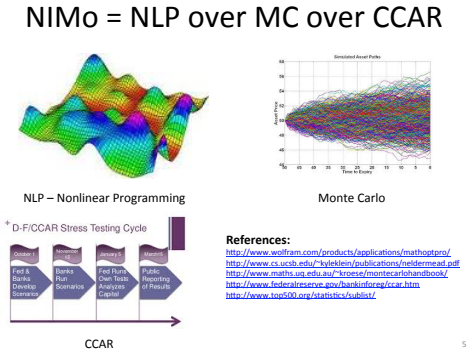

Figure 1: NIMo Automation - CCAR feeds Monte Carlo and run through Non Linear Optimizer

LCR, NSFR, GSIB, TLAC, and CCAR preparations and rollouts through the next several years will be quite challenging to existing bank systems. There are GSIBs working on significant balance sheet system speed-ups just to control hardware costs. Some of the system speed-ups are mandatory simply to meet static simulation regulatory reporting requirements. We will maintain AWS NIMo Reference Servers for Firms to check their internal balance sheet simulation results against standard product models. Firm's are welcome to use the reference server code internally, optimized to their production environment. We will also provide code and expertise to GSIBs, banks, financial service firms, and the FRB to assist in optimized simulation. Up-to-date competitive balance sheet simulation tools on commodity microprocessors should promote safety in the US Banking system. 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 each Firm's capital planning and regulatory reporting, obviously remain wholly controlled by the respective Firm. The long-term goal is to automate the management of revenue margin (NIM) by adjusting business mix, product pricing, and funding through security/contract level

Monte Carlo simulation and contemporary optimization algorithms assisting Firm Management.

The reason Firms work with the Consortium is raw simulation performance applied to their balance sheet. In 2016 the NIMo Reference Server runs a static worst-case 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, a few tenths of a second. That is important if there is need to run this balance sheet simulation in a Monte Carlo expected case framework. It is also important for:

- Internal Audit and Model Review
- Development Testing
- Market and Credit Risk Review
- External Regulatory Reporting
- Firm Management & Asset/Liability Committee Reporting

If a bank needs that kind of world-class competitive performance they run it with their internal High Performance Computing groups or work with the Consortium. By tuning the code to the compiler and the underlying hardware, we take advantage of Multi-core, Fused Multiply-Add, and SIMD instructions on superscalar pipelined commodity processors (see Figure 2).

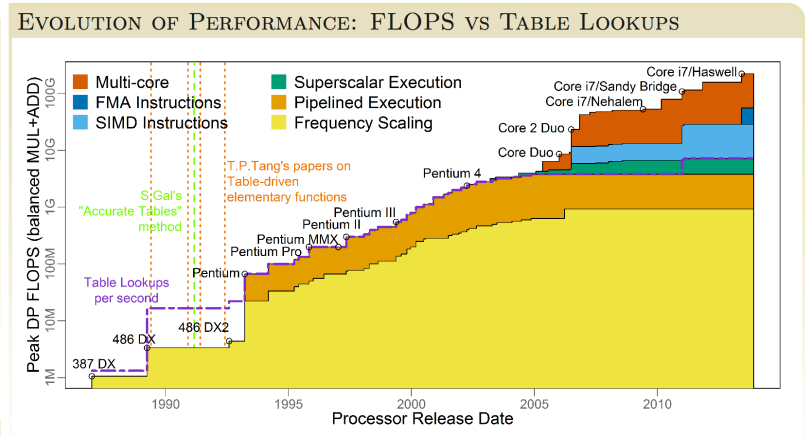

Figure 2: Recent Commodity x86 Microprocessor Performance Improvements (Dukhan, HotChips)

The Consortium uses commodity x86 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 NIMo simulation code, the entire annual capital allocation plan for a GSIB 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 (or custom multifactor) 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 historic/current/forward market data or scenario 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 been in development 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. But without direct access to the accrual portfolio inventory that's like optimizing your trillion dollar MBS portfolio, flying blind to new business, and having no access to a prepayment model. It's colorfully called "*steering a rocket with a joystick*." 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 implausible at any of the large CCAR banks, although the data and now the computational capacity are available. There are billions of dollars in annual revenue on the table for automating large balance sheets.

If the accrual portfolio inventory balances and rates of return get reported daily (or even weekly) there is typically enough computational power available in the existing Firm 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.

From an investor, supervisory, or regulatory standpoint there are several different applications of NIMo. Banks both large and small can be ranked on the realized efficiency of their funding of their respective balance sheets. For a given bank the accrual portfolio and capital allocation plan is known at the start of the fiscal year. The NIM, the accrual portfolio, and historical market and econometric variable levels are known at the end of the year. You do not necessarily know each bank's balance, rate of return, and default model for each of their products – but you can use standard proxy models for all the products across all the banks, given some error tolerance. In fact, you could fit standard linear and non-linear regressions to the US Federal Reserve's aggregate accrual portfolio security/contract data and use that as the standard balance and rate model. Using standard product models for all banks makes the end of year efficiency number comparable. Simulating the optimal implementation of the capital plan from the beginning of the year and taking the ratio of the Day-Zero-Optimized-NIM/Realized NIM gives an estimate of the bank's operational efficiency and market dexterity. If the bank's native information asymmetry doesn't assure that Realized NIM > Day-Zero-Optimized-NIM there is something wrong. Investors can compute the relative value of banking acquisitions on a net accrual portfolio basis. Regulators can simulate the aggregate USD accrual portfolio with the optimal aggregate capital allocation plan to provide early warnings of operational impairments. The optimal aggregate capital allocation plan may be quite different than the sum of the realized capital allocation plans. The regulator knows what each of the banks are reporting as well as the aggregate picture of the current and expected performance. Who says no?

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 early 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 straightforward 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.

Our current estimate is a trillion dollar balance sheet NIMo Monte Carlo expected case simulation will require several hours on upwards of 100 cores. If you do not get the single core balance sheet code optimized performance 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 code optimization of the Monte Carlo simulation 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 x86 core.

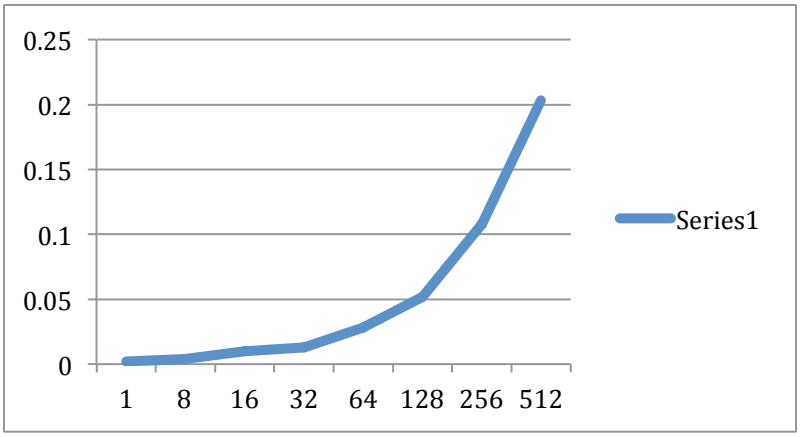

Figure 1: Balance Sheet Hold Flat Simulation seconds vs. thousands of accounts

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.

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 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. Note we are coercing the parameters of the problem to be powers of 2 so we can stay quad word aligned for later optimization. There is about a factor of ~25 left to optimize away from the reference implementation on a single core of my old iMac.

The following references cover aspects of optimizing codes implementing memset()and memcpy() functionality. They basically tell you step by step how to optimize the hold-flat balance sheet simulation.

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 microprocessor's FMA FP execution units are totally idle in this Hold Flat benchmark, it is easy to slip in code for valuing regression fits without changing he performance all that much (we will review that next in Figure 4); 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 desktop microprocessor.

```
    for (int i = 0; i < max_acct; i++ ) {
        this->balance[0] = balance[i];
        for (int j = 1; j < max_periods; j++ ) {
            *(this->balance + i*this->num_cols  + j)=  *(this->balance +
i*this->num_cols  + j-1)* coeff[j] * log(mkt[j]/mkt[j-1]) + exp(c[j]) * mkt[j-
1];
        }
    }
    for (int i = 0; i < max_acct; i++ ){
        this->rate[0] = rate[i];
        for (int j = 1; j < max_periods; j++ ) {
            *(this->rate + i*this->num_cols  + j)=  pow(*(this->rate + i*this-
>num_cols  + j-1), 0.99);
        }
    }
```

Figure 4: Balance Sheet full regression fit model C++ Reference Code

To get a sense of how the worst-case static simulator performance will vary when regression models are introduced, to improve on the simulation fidelity of the hold flat model in Figure 3, we show sample regression code in Figure 4. This benchmark code runs 512K accounts and 60 months forward in 0.76 seconds on the same single x86 core as the Figure 3 example. The Hold Flat execution time was 0.2 seconds and the sample regression kicks up the execution time by a factor of about 4x. Again the code compilation for this benchmark is mostly unoptimized. The time consuming portions are the evaluation of the scalar log(), pow(), and exp() intrinsic math functions, the divide mkt[j]/mkt[j-1], and the very light cache pressure introduced having to source parameters (coeff, c, mkt) for the expression evaluation. There are distortions in this performance benchmark to note. The performance benchmark does much more aggregate computation than many production CCAR simulators I have seen running but the L2/L1 cache pressure of the benchmark is much lighter than the prod CCAR executions I have examined. Like the Hold Flat simulation, this code can be substantially optimized using the right compiler and a vectorized intrinsics math library like MKL or MASS. It would be surprising if optimized, this code ran to completion in more than 0.05 seconds for a full balance sheet simulation on a single core of my old iMac. So, optimized you could run a thousand different balance sheet scenarios in under a minute on a single desktop core. The runtime is so small there no real need to think about further parallelization and the code is dirt

simple and it is free on the internet. That may be a CCAR performance improvement for some banks.

The complication as you look at real balance and rates models is that GSIBs really want to execute the computation as shown in Figure 5. Those functions F1() and F2() are typically more than some simple regression fit and they may have more required parameters than displayed in Figure 5. Moreover there could be thousands or tens of thousands of such functions required for a full balance sheet simulation. This will put some pressure on the memory hierarchy feeding the execution units but it seems tractable.

```
for (int i = 0; i < max_acct; i++ ) {
        this->balance[0] = balance[i];
        for (int j = 1; j < max_periods; j++ ) {
            *(this->balance + i*this->num_cols  + j)=  F1((this->balance +
i*this->num_cols  + j-1), coeff[j], mkt[j], c[j]);
        }
    }
    for (int i = 0; i < max_acct; i++ ){
        this->rate[0] = rate[i];
        for (int j = 1; j < max_periods; j++ ) {
            *(this->rate + i*this->num_cols  + j)=  F2((this->rate + i*this-
>num_cols  + j-1), coeff[j], mkt[j], c[j]);
        }
    }
```

Figure 5: Balance Sheet full model fit model C++ Reference Code

The practical difficulty is with manually optimizing 10K Fi() functions. You can select benchmark Fi() functions, optimize the code, and get good performance numbers. If the Fi() functions stable it is easier, I expect there will be a reasonable variation in the function code for the near term as this quantitative modeling effort matures. You can always simply run them all and then just optimize the longest running Fi().

There are some obvious implications for applications such as:

1. Static worst case balance sheet scenario simulation
2. Expected case balance sheet simulation, and
3. Net Interest Margin optimization.

Static worst-case balance sheet simulation with regression fit models:

Application – simulate current balance sheet on one scenario forward 60 months.

- There is no more than a few seconds of unoptimized computation to execute for a given scenario on a single core of a generic x86 commodity microprocessor.

- If there was demand, you probably could get a complete balance sheet simulation 512K accounts x 60 months running under 10 seconds on an old laptop.
- If your infrastructure costs running CCAR balance sheet simulations are much greater than the cost of a good laptop – you might have a 2016 cost savings opportunity.

Expected case simulation:

Application – simulate balance sheet on randomly generated LMM MC scenarios

The desire to predict accurately when funds will be available will spur the transition from linear regression models to full balance, rate, and default models for cards, deposits, loans. Recall banks generate revenue by crossing assets and liabilities at the Net Interest Margin. The more accurately you know when excess funds become available, then the less funding your capital plan acquisitions costs, and thus the greater your Net Interest Margin. We may or may not be able to forecast the future market better than the next Firm, but we can dynamically minimize our new business funding cost, subject to market and operational expectations. That is a big deal.

Models for individual cards, deposit, or loan agreements, although computationally feasible, seems unlikely in the short term. Individual contracts with lots of optional draw downs/prepayments, incentives, and fees but lacking standard security indicatives like a maturity date are complicated to simulate accurately with standard Front Office quantitative models. One interesting point that could be pursued is the modeling relationship between one customer's deposit account, their credit cards, their car loans, and their mortgage. There can only be several hundred million USD deposit accounts, call it a billion. Same with cards (see http://www.creditcards.com/credit-card-news/ownership-statistics-charts-1276.php). Mortgages represent way fewer contracts, under 20 mm according to the Fed (http://www.federalreserve.gov/econresdata/releases/mortoutstand/current.htm) . Does not seem out of the question technically that you could simply directly model the customer's daily cash flow and just dynamically correct for individual mortgage prepayments, default judgments, loan line draw downs. Maybe you can do this in some Fintech startup? Technically it seems feasible.

The local pooling of cards, deposits, and loans seems more likely from a Front Office quantitative perspective, since it makes these products look like MBS and bundle new business operations modeling along with roll-off inventory rates and credit modeling. Each local pool will be considered a distinct separate parameterized analytical security. Pools on the order of one hundred contracts seems to leave large banks with accrual inventory of no more than 5 to 10 mm positions. These 10mm positions feed aggregated balances and returns to max 500K accounts. Each of the

account balances, rates, and defaults needs to be simulated forward 60 months for several thousand Monte Carlo paths/scenarios to derive an average balance, rate, and default (could move to weekly or daily simulation periods depending on the underlying model accuracy). There is significant computational capacity currently available to execute these models. They will be highly protected, proprietary models. Seems like you will need 10K to 100K Monte Carlo paths to converge (presumably even with low-discrepancy quasi-MC algorithms), optimize your code accordingly and do not parallelize prematurely. The NIMo Monte Carlo has regular, balanced, and predictable data access patterns and instruction flow so it will give you tolerable Amdahl's Law behavior (see http://arxiv.org/pdf/1507.04383v1.pdf) the main problems to monitor on the Monte Carlo will be the Amdahl's law limits for scaling, the L1 and L2 cache performance for the security modeling, and clock speed of the microprocessor (can you get a bunch of 4+ GHz Skylake or better or do you have to downshift the clock for something like Intel Phi?).

NIMo Optimization:

Application – simulate new business over the expected case balance sheet and optimize the new investment to maximize the NIM.

If you can control the approximation error in computing the expected balance, return, and default for both the roll-off inventory and a parameterized set of new investments then you can optimize the selection and timing of spending and new investments in the capital plan. The optimization works through a multiperiod combinatorial NLP selecting new investments across Operations, Rates, and Credit expectations and constraints to maximize the Net Interest Margin. The super cubic asymptotic runtime complexity of the combinatorial NLP is not an insurmountable barrier since the cardinality of the new investments is small enough (est. around 10K) to not require massive hardware. To a large extent the NLP runtime is independent of the cardinality of the positions in the roll-off portfolio. You simply need to know the timing of the availability of excess funds rather than the number of cards contracts or mortgages.

In order to make the NLP not simply converge to the new investments with the largest interest rate we make the multiperiod optimization depend on both rates and credit as well as aggregate sensitivity constraints.

Making the optimization depend on both operations and market expections keeps convergence away from units with a history of missing or being late in hitting capital plan targets. We can factor in fraud history to our capital plan optimization.

At a certain point, we can use the optimizer to compute the Firm revenue forgone due to bad or late accrual inventory data reporting or lagging security modeling accuracy.

Global Balance sheets are likely to be decomposed into several independent optimization subproblems corresponding to tax jurisdictions. Presumably, I cannot easily (e.g., without cost) repatriate funds from deposits in Spain to cards in Singapore. We will run dynamic programming on the subproblems to get the global optimial Net Interest Margin.

The final step in the automation of the Firm Capital Allocation process we will take real-time feedback from the market and operations to adjust the optimizer goals to dynamic inputs. It is an example of applying Control Theory to Balance Sheet management.

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. 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
- 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.

current_market: Market and macroeconomic variable levels for the user selected as_of date. Note this server does nothing much more than take user variable names and levels for substitution into the balance, rate, and default models. The server just simulates balance sheets. It does not discount cashflows to compute risk. It just does one thing really fast. That one thing is simulating balance sheets. User supplies the market data, the econometric data, the scenario data and the reference server takes them as variable names and levels for substitution in the models. Otherwise the reference server is purposely oblivious to the meaning of these quantities. This is not a Risk server.

- Market and macroeconomic levels required for balance, rate, and default model evaluation.
- Restricted to USD initially – bur server will convert FX given exchange rates.
- 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.

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 macroeconomic variable levels required evaluate balance, rate, and default models. We do not cook market data, the user does that before calling the reference server.

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. **initialize_balance_sheet** – compute the initial balances, return rates, realized defaults for the first time period of the simulation.
3. **simulate_balance_sheet** - Simulate the evolution of the balance sheet over a simulated passage of time using either static or stochastic market/macro scenarios.
4. **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.
- 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 and 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. initialize balance sheet
   f. simulate balance sheet
   g. 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. set nimo configuration – full simulation base
   e. initialize balance sheet
   f. simulate balance sheet
   g. 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. initialize balance sheet
   g. simulate balance sheet
   h. 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. initialize balance sheet
   h. simulate balance sheet
   i. optimize capital allocation plan
   j. get balance sheet
   k. get capital plan