2016 Enterprise Risk Management Symposium

April 6–8, 2016, Arlington, Virginia

Many-Task Computing Brings High Performance and Simplicity Into Principle-Based Applications

By Huina Chen and Henry Bequet

Many-Task Computing Brings High Performance and Simplicity Into Principle-Based Applications

Huina Chen, FSA, CERA,[1] and Henry Bequet[2]

Abstract

Insurance regulations are undergoing a paradigm shift in determining capital and reserves. With Solvency II in Europe and principle-based reserving in the United States, the old formula-based static approach is being replaced or supplemented by a principle-based dynamic method. This change requires enhanced modeling, stochastic simulation and sensitivity analysis, which pose challenges to insurers whose infrastructure has not been built for big computation. This paper introduces a method of building complicated models with small reusable modules and running them on a many-task computing platform to achieve high performance with simplicity. The paper presents an example of a term life insurance model built to take advantage of computer hardware for parallel computing at the task level.

1. Introduction

The principle-based regulations, such as Solvency II (EIOPA 2014) in Europe and principle-based reserving (PBR) (AAA 2014) in the United States, compared with traditional formula-based regulations, put more emphasis on tail risks in insurance contracts. Tail risks are possible events that yield high impacts with low probability. They can send an otherwise financially sound institution to insolvency. Solvency II requires sufficient capital to be held such that there is only a one-in-200 chance an insurer is not able to cover the liabilities. Under PBR, the stochastic reserve is set as the average of the highest 30 percent reserves calculated for multiple scenarios. Under the principle-based framework, it is not uncommon to calculate reserves or capitals using thousands of scenarios per risk factor. In some cases when stochastic-on-stochastic calculation is involved, it may take days, or even weeks, of computation to generate results for reporting. The insurance industry has been searching for various ways of improving computation performance. Up to a few years ago, focus was on faster central processing unit (CPU) clock speeds to increase computational performances. Today, the focus is on increasing the number of cores or threads of execution in computer architecture. Insurers have channeled investments in hardware, especially computer grids with multiple engines, to shorten the total runtime. This, in turn, requires changes in the structure of analytical programs to leverage the power of more threads. The transition to multi-thread has been problematic so far because parallel programming is notoriously hard for all practitioners, from software developers to risk analysts including actuaries. The fact that most

[1] Huina Chen is a principal research statistician developer at SAS Institute and can be reached at huina.chen@sas.com.

[2] Henry Bequet is a director of development at SAS Institute and can be reached at henry.bequet@sas.com.

fourth-generation programming languages (4GLs) used by risk professionals typically do not support parallel programming very well has made matters worse.

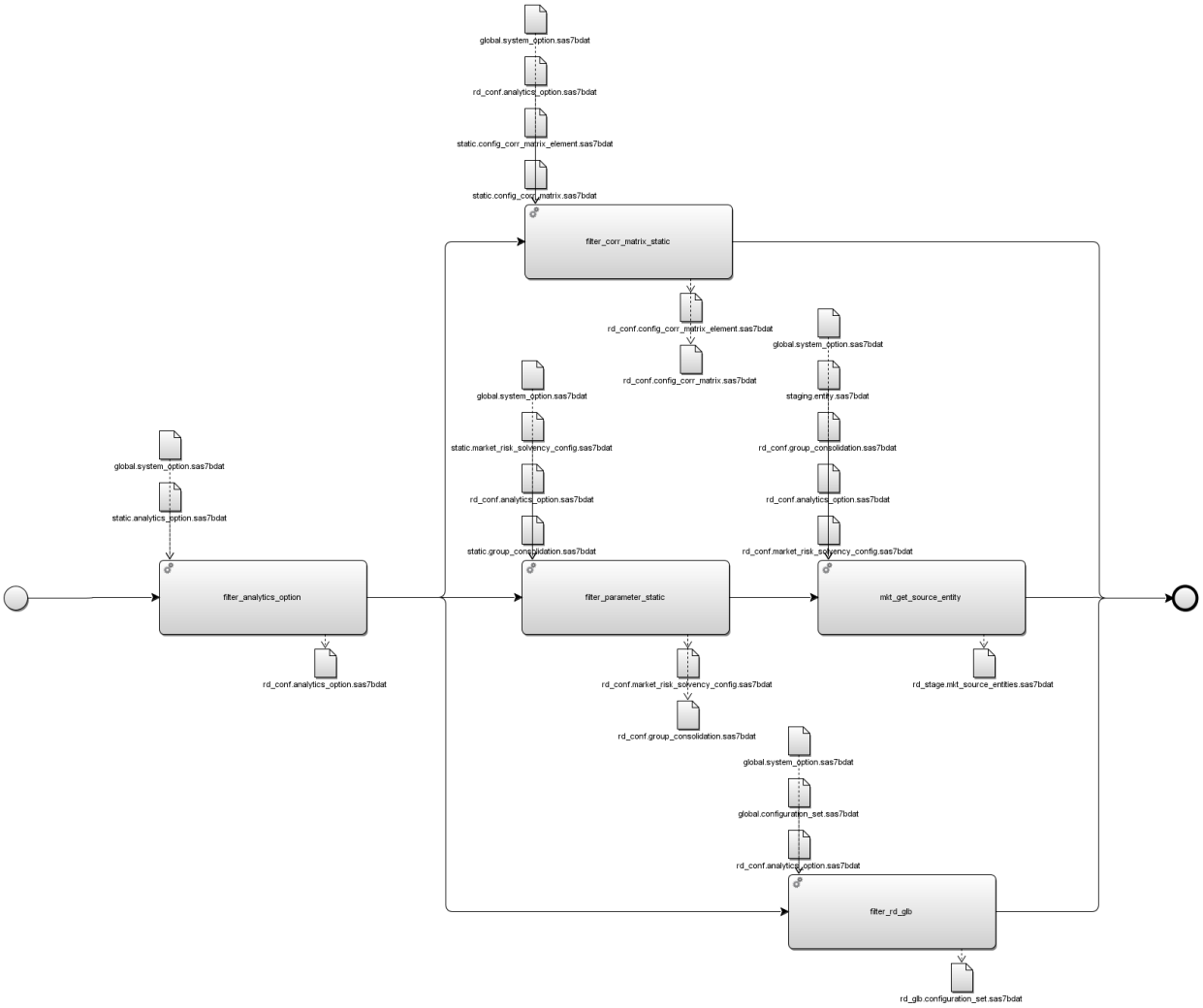
This paper provides a methodology that promises to leverage the power of many threads while keeping the development of models to a manageable level of complexity. Many-task computing offers risk analysts a simple way to achieve the high performance computing required in principle-based applications.

2. Many-Task Computing

Many-task computing (MTC) is an approach to parallel computing that puts the emphasis on data synchronization rather than code synchronization (Raicu, Foster and Zhao 2008). In MTC, parallel programs are organized as sequences of computations called job flows. The main point of a job flow is to transform data through a set of tasks that perform calculations on a set of inputs to produce a set of outputs. Once all the inputs of a task are available, the task can be dispatched to a thread of execution independently of other tasks. The fact that the parallel scheduling of tasks is decoupled from the analytical code gives MTC its ease of use and performance.

Another benefit of job flows is that they convey the overall structure of the parallel program in a graphical manner. For instance, the job flow in Figure 1 performs some analytical computing on SAS datasets.

Figure 1. Sample Job Flow in Graph



The parallel nature of the computations is readily visible to the end users.

The job flows can be created via a graphical user interface (GUI) in which end users can drag and drop tasks and their associated data. The MTC platform automatically figures out the sequence of execution and the parallelism, so there is no need for the risk analysts to synchronize the computations.

The job flows can also be defined programmatically, such as with the following example in Python, which defines the job flow in Figure 1.

```
def test_simple_flow(self):
  # We get a context for a flow definition (name)
  definition_name = 'testDefinition_'+ str(datetime.now().timestamp()).replace('.', '_')
  ic = Context(definition_name)

  # We add analytical tasks to the flow
  filter_analytics_option(ic)
  filter_rd_glb(ic)
  filter_corr_matrix_static(ic)
```

```
mkt_get_source_entity(ic)
filter_parameter_static(ic)

# We run the flow
ic.run()
```

Note that in the Python example, the computations are given to the system out of sequence, but here again, the system automatically figures out the order and the parallelism of the tasks.

3. Model Design

An MTC platform enables modelers to design contents without thinking about how to couple the model with the hardware to achieve parallelism. This empowers risk analysts, such as actuaries, who possesses insurance domain knowledge but often lacks computer science background, to develop models with ease. We only remind modelers of the following three commonly used model design principles to take full advantage of hardware, such as computer CPUs and memories.

Principle 1. Use vectors to replace loops whenever applicable. When evaluating a block of policies under multiple scenarios, a computation procedure can loop through each scenario and, inside each scenario loop, through each policy. Modern computer languages such as SAS and Matlab provide efficient vector operations, which are generally much faster than executing loops. Grouping policies and/or scenarios into blocks and using vector operations to calculate each block as a whole significantly increase speed. The size of blocks depends on the purpose of the model and the hardware specification.

Principle 2. Separate logic codes from data. Many applications, such as simulation, execute the same logic on different data. Compiling the same piece of codes once and using them multiple times save computing resources. It also facilitates code maintenance and helps mitigating modeling error.

Principle 3. Design the model as a collection of small tasks instead of a big indivisible program. Single-threaded programs, which are designed to run on one single thread in one computer, usually take all input data, calculate many steps sequentially and then output the results. Such a “one big task” approach does not fully utilize CPU power in a computer or a grid of computers with multiple threads. Alternatively, the model can sometimes be divided into many small tasks. There are four advantages in structuring a program as a collection of small tasks, as long as they are in the right order.

1. It enables parallelism at the task level. When an MTC platform is available, the independent tasks can run concurrently as long as there are multiple computer threads available.
2. It is easier to execute an entire small task in computer memory, therefore reducing I/O operations and increasing speed.
3. It increases traceability. Debugging a model often requires more time than what is spent on developing the initial model. Small tasks provide explicit breaking points for intermediate results. Because each small task explicitly defines its input and output, when a final result is different from expectation, these intermediate results become very helpful in identifying the issues. This is also a desirable feature for auditors to verify the data and processes.

4. Models are easier to maintain and upgrade. In the event that regulations for certain aspects (for example, reserve calculation) are changed, codes for other tasks (for example, premium and death benefit projection in a pricing model) can be reused without change.

4. Example

In the following, we use a stochastic pricing model for level term life insurance products to illustrate how a well-designed actuarial model running on an MTC platform can help insurers easily achieve high performance.

A level term life insurance contract guarantees level premiums and level death benefits over a term (a prespecified period of years). The premiums and death benefits can vary if a policy stays in force post term. It is the simplest, therefore least expensive, life insurance product. We choose the level term life pricing model as an example for two reasons.

1. The product enjoys significant market shares.
2. The model is simple yet complete as it covers core insurance underwriting risks and their related cash flow projection.

A simple model makes it easier for readers to understand the principles of good model design in order to take full advantage of computer hardware to achieve speed. More importantly, we want to discuss how to easily create new models for more complicated products like universal life products, by using/modifying existing task nodes or adding new nodes to the simple model. Because the focus of this paper is on model design, we disguise real data while providing a description of data used in the model. Refer to Atkinson and Dallas (2000) for more details on life insurance products and formulas.

4.1. Model Data

4.1.1. Policy Data

The portfolio consists of 1,098 model cells representing 1 million level term life policies with a specified pricing distribution. Each model cell is uniquely defined by a combination of policy characteristics such as term, issue age, gender, risk class and policy size band.

4.1.2. Scenario Data

Mortality and lapse are the two major risk factors that greatly impact the profitability of a term life business. An insurer can run into financial distress if the actual experience of either factor adversely deviates too much from the best estimates. A projection model, which captures the variation of mortality and lapse, can provide insights for insurers into the potential risk distribution due to the two policyholder decrements. The level term life model we present here does cash flow projection for one or multiple mortality/lapse combined scenarios. A mortality scenario is a matrix of mortality multipliers to be applied against the base mortality rates. We apply multipliers that reflect possible variation in mortality experience. They vary by projection year and term length. They remain the same for different

polices with the same term length during the same projection year. A lapse scenario has the same structure as a mortality scenario. The corresponding multipliers are factors to be applied against the base lapse rates. See Clark and Runchey (2008) for details of how to generate the stochastic mortality and lapse scenarios. We run 1 million stochastic mortality/lapse combined scenarios to test the model's speed and scalability.

4.1.3. Output Data

The model generates 1 million aggregate 30-year income statements for the block of business for 1 million mortality/lapse combined scenarios.

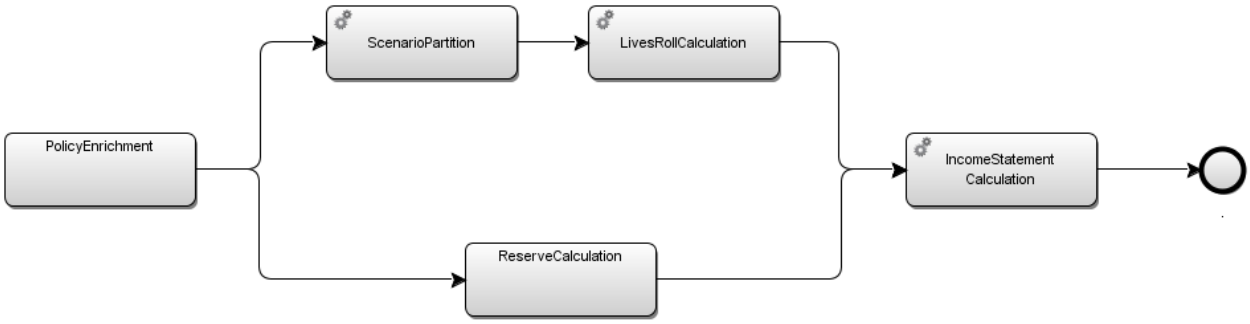
4.2. Runtime Environment

The model is written in SAS language and runs on SAS workspace servers. The MTC platform is tested on a testing computer with a 1TB RAM, 80 hyper-threaded cores at 3.20 GHz.

4.3. Modeling Steps

For each model cell, the model starts with one unit (1,000 face amount) at issue, takes a mortality/lapse combined scenario to compute the number of deaths, lapses and in-force lives for each projection month, determines unit statutory reserves, and then calculates cell-level income statement for 30 years. The model further rolls up cell-level income statements into an aggregate income statement for the block of business for each scenario. At the end, 1 million 30-year aggregate income statements are produced. Figure 2 describes the high-level job flow.

Figure 2. Job Flow for a Term Life Insurance Pricing Model



The boxes with wheels at the top left corner represent task nodes. A task node is a calculation unit. It has input and output tables as shown in Figure 1. We disguise the input and output tables for task nodes on the job flow figures from now to the end of the paper to increase readability. The boxes without the wheels represent sub-flows. A sub-flow is a collection of related task nodes grouped in a certain order. There can be as many layers of sub-flows as needed. One can always drill down a sub-flow to see the tasks or sub-flows it includes. Sub-flows increase flow readability and help readers to understand the flow logic. However, they do not contribute to calculation. Task nodes are the only calculation units.

Step 1. Enrich policies to link policies with assumption and product data. It is conducted in sub-flow *PolicyEnrichment*. This step extracts data from various input tables so that each policy gets its

corresponding assumptions and product information such as mortality rates and premium rates. According to Model Design Principle 1, we treat the 1,098 model cells as a block and calculate them as a whole using column operations in SAS. We do this for two reasons.

1. It is easier to compute the aggregate result for the block of business when packing all model cells into one block.
2. The SAS language treats columns as variables and does efficient variable operations, such as addition, subtraction, multiplication, division and comparison, among columns.

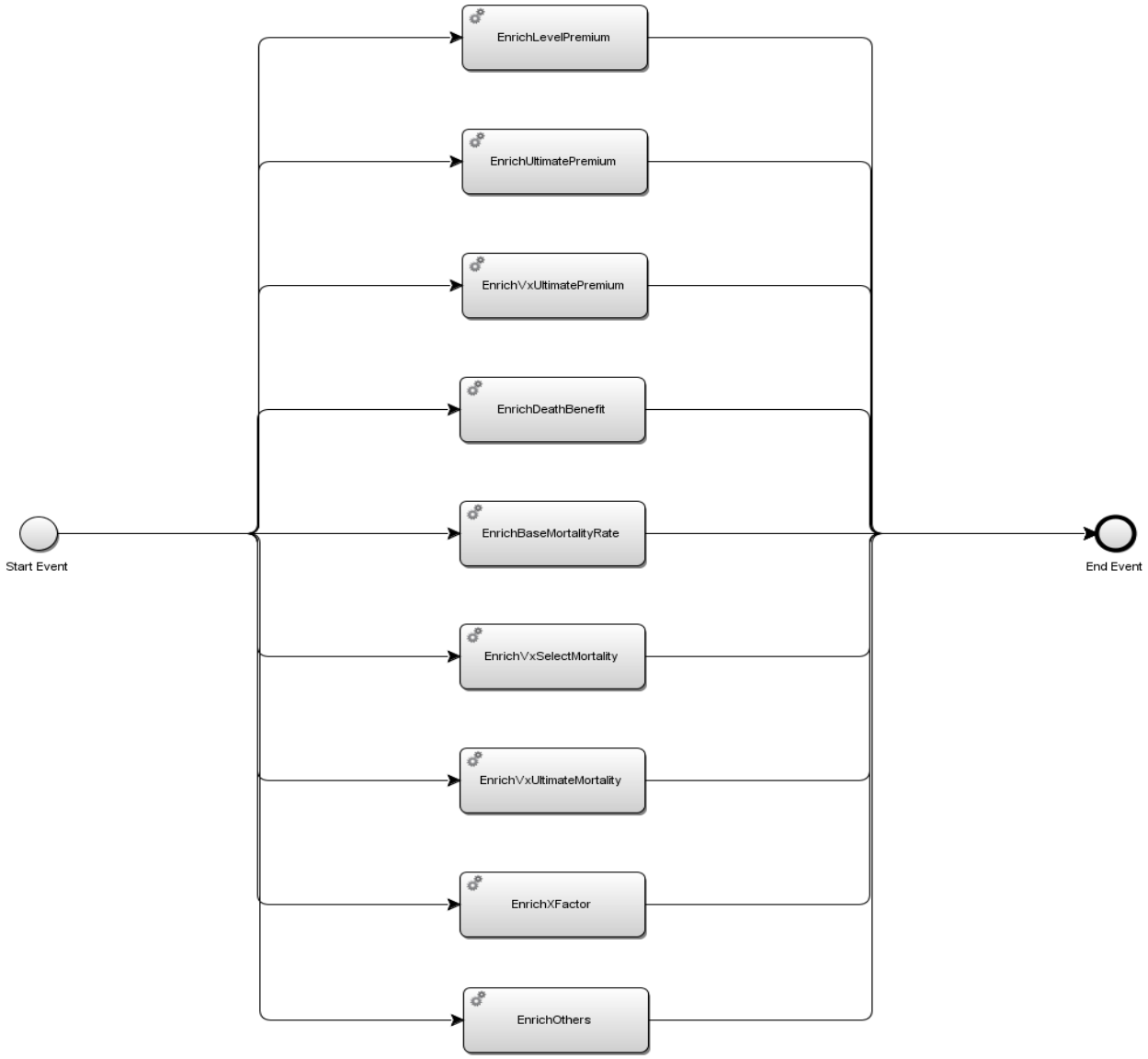
Reflecting our earlier discussion of Principle 1 in Section 3, such operations are much faster to execute than using loops.

For example, we use an SAS data table, which stores policy keys, their corresponding base mortality rates and mortality multipliers, to calculate the modeling mortality rate for each policy. Formula (1) treats all rows in the table as a whole and calculates mortality rates for each row in the table.

$$\text{Mortality\_Rate} = \text{Base\_Mortality\_Rate} * \text{Mortality\_Multiplier} \quad (1)$$

The policy enrichment step involves many data movements and may take much time. The assumption and product data are related to policy data but do not depend on each other. According to Model Design Principle 3, we break down the enrichment tasks into small ones so they can run in parallel when possible. Figure 3 shows the enrichment tasks.

Figure 3. Policy Enrichment Sub-Flow



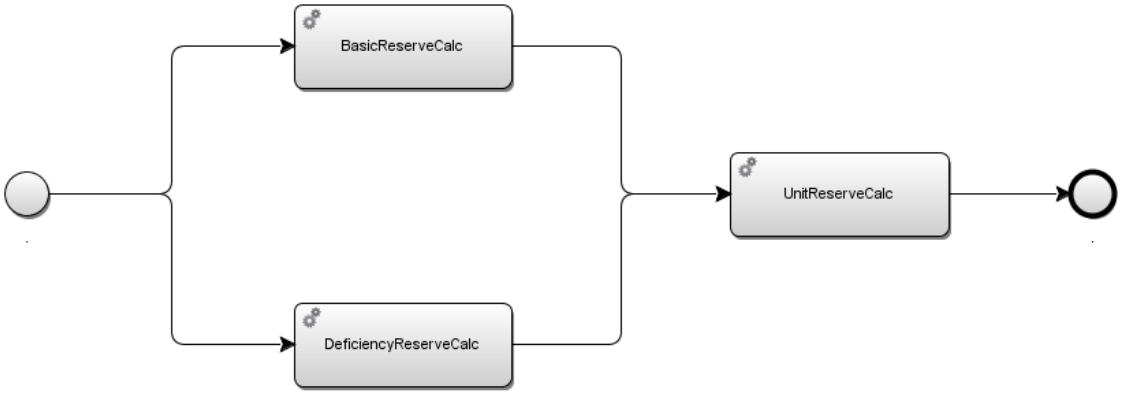
Step 2. Partition scenarios. Task node *ScenarioPartition* splits stochastic mortality/lapse combined scenarios into blocks. The blocks of scenarios are then fed to Step 3 to calculate lives roll. Either the MTC platform or a modeler determines the number of scenarios inside each block, taking into account computation efficiency. After testing a range of block sizes, we found 10 scenarios per block lead to best performance in our testing environment described in Section 4.2. Therefore, we split the 1 million scenarios into 100,000 blocks. This task node has one table containing 1 million scenarios as an input, and 100,000 tables each containing 10 scenarios as outputs. To facilitate model design, the MTC platform treats the 100,000 tables as a collection. A modeler only has to define the tables as a partitioned table with one table name to represent the collection.

Step 3. Calculate lives roll. Task node *LivesRollCalculation* calculates how many lives of death, lapse and in-force for each projection month. It applies corresponding multipliers to each policy's base mortality

rates and base lapse rates in different scenarios to obtain modeling mortality and lapse rates, and then projects monthly policy statuses. The same lives roll calculation code repeatedly processes 1 million scenarios to generate 1 million lives roll projection. We apply Model Design Principle 2 to separate calculation logic from data, and then follow Principle 3 to create small tasks for parallel execution. The task takes the partitioned scenario table generated from Step 2 as an input, runs 100,000 partitioned tasks, and then produces a partitioned lives roll table (a collection of 100,000 tables) as the output. Notice that even though there are 100,000 tasks, the job flow only shows one task node, *LivesRollCalculation*. This is because the MTC platform is designed to automate task scheduling. As long as a modeler defines the task node as a partitioned node, the MTC platform automatically provides multiple threads to run the partitioned tasks on the background. This makes model design easy and clean. Modelers can focus on content instead of the underlying execution.

Step 4. Calculate unit statutory reserves. This is done in sub-flow *ReserveCalculation*. We can see from Figure 2 that this sub-flow runs independently from the *ScenarioPartition* and *LivesRollCalculation* tasks. Because unit reserves use prescribed mortality and lapse assumptions differently from what are used in pricing, pricing scenarios and lives roll projections do not affect unit reserve calculation. Figure 4 shows the composition of the sub-flow. Basic reserve and deficiency reserve are calculated in parallel. They rely on different mortality assumptions and premium rates while using the same piece of computation logic. We again apply Model Design Principle 2 to call the same compiled codes for each of them.

Figure 4. Unit Reserve Calculation Sub-Flow



Step 5. Calculate income statements. Task node *IncomeStatementCalculation* calculates premiums, death benefits, reserve increase, investment income, expenses, tax and distributable earnings to form cell-level income statements for 30 years. The model then multiplies each cell's per-unit-issued results with the number of units defined in the pricing distribution to generate gross results for each cell. In the end, it sums up the gross results from all cells to create the 30-year aggregate income statement for the entire block of business. The task takes the partitioned lives roll table and non-partitioned reserve table as inputs, runs 100,000 partitioned tasks, and generates a partitioned aggregate income statement table (a collection of 100,000 small tables) as the output. A modeler can choose to combine the collection of small tables into one big table or leave them as they are.

The model produces 1 million aggregate income statements for 1 million mortality/lapse combined scenarios. A modeler can add additional task nodes to slice and dice the results for more insights into the portfolio. For example, one can take the present values of each aggregate income statement's line item and form a distribution for each line item to see their risk characteristics. A modeler can also add a cell-level income statement as another output partitioned table of Step 5, then add another node to summarize income statement items by policy characteristics, such as gender, age, risk class, product size and term. This can provide useful information on what type of policies are more profitable than others.

4.4. Performance

We ran a range of scenarios to test the performance. Table 1 shows the performance results of our SAS model running on the 80-thread MTC platform.

Table 1. Performance Results

Number of Scenarios	Run Time Total	Run Time (in seconds) per Scenario per Thread
1	3 seconds	3.00
1,000	35 seconds	2.81
1,600	42 seconds	2.10
1 million	6 hours and 55 minutes	1.99

We compare our model performance with a single-threaded Excel model. Both models use the same input data, do the same calculation and generate the same output data for any one of the mortality/lapse combined scenarios. For a single scenario, the Excel model runs for 88 seconds, while our SAS model based on MTC runs only 3 seconds. Because Excel is single threaded, it has to use Visual Basics for Applications (VBA) to loop through all 1,098 model cells, then do aggregation. The only parallel computing Excel can achieve is to distribute scenarios to computer grids. It would take over 12 days for the Excel model to generate 1 million scenario results were it to run on a grid with 80 cores, compared to the seven hours our model needs.

4.5. Varying models

Because an MTC model is composed of a collection of small tasks, it is convenient for a modeler to create new models by adding, replacing or modifying some nodes of an existing model. For example, one can take the term life pricing model described in Figure 2 and turn it into a universal life (UL) pricing model. Both models need to link polices with assumption and product data. Sub-flow *PolicyEnrichment* can be modified to link UL-related assumption and product data to policies. For example, we can replace task node *LinkLevelPremium* with task node *LinkTargetPremium*. Many formulas for reserve calculation and income statements also need to be changed for UL products. Because interest rate risk is critical to UL products, a modeler may want to do stochastic simulation on the interest rate instead of mortality and base lapse. Therefore we use task node *ScenarioPartition* to partition interest rate scenarios. Furthermore, the codes for task node *LivesRollCalculation* should be modified to take into account dynamic lapse, which is directly affected by interest rates.

5. Summary and Future Work

We have shown that models built on the concept of MTC and run on an MTC platform can significantly reduce runtime to meet the big computation challenge faced by principle-based applications. The example illustrated in Section 4 demonstrates a 40 times performance improvement of our model, compared to Excel.

To bring computing performance to a higher order of magnitude, we are looking into using graphics processing units (GPUs) to further reduce runtime with lower costs (Grauer-Gray et al. 2013). A GPU is a programmable logic chip traditionally used to process graphical data in animation, visual effects and video editing. In most computers, the central processing unit (CPU) handles the majority of the processing and farms out graphical processing to one or more GPUs. A key difference between CPU and GPU is that a GPU handles thousands of threads of executions of the same program (e.g., the transition of an animation frame) on different data (e.g., tiles in the frame) while a CPU handles tens of threads of executions. In other words, it is possible to leverage the many threads of a GPU when a task requires the same computation repeated for a variety of data. Several problems in risk management, including stochastic simulations, are a good fit for GPU architecture. Alas, the coding aspects of GPUs are nontrivial. Therefore acceptance of using GPUs for analytics has been rather slow. We are working on delivering components of job flows that automatically harness the many threads of GPUs and hide the coding complexities of GPU programming. We believe such a high-level tool can put the power of GPUs within the grasp of risk analysts so that they can easily harness thousands of threads of execution.

If a GPU chip with 5,000 threads is used for the MTC platform, we estimate the runtime of the model described in Section 4 can be further reduced from seven hours to less than half an hour for 1 million scenarios. Because a GPU chip with thousands of threads only costs thousands of dollars, we believe this represents the future of high performance computing in principle-based modeling.

References

American Academy of Actuaries (AAA) Life Principle-Based Approach Practice Note Work Group. 2014. "Life Principle-Based Reserves Under VM-20." Exposure draft.

Atkinson, David B., and James W. Dallas. 2000. *Life Insurance Products and Finance: Charting a Clear Course*. Schaumburg, IL: Society of Actuaries.

Clark, Matthew, and Chad Runchey. 2008. "Stochastic Analysis of Long-Term Multiple-Decrement Contracts." *Actuarial Practice Forum* (July).

European Insurance and Occupational Pensions Authority (EIOPA). 2014. "Technical Specification for the Preparatory Phase, Part 1." EIOPA-14/209.

Grauer-Gray, Scott, William Killian, Robert Searles, and John Cavazos. 2013. “Accelerating Financial Applications on the GPU.” *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units* GPGPU-6: 127–36.

Raicu, Ioan, Ian T. Foster, and Yong Zhao. 2008. “Many-Task Computing for Grids and Supercomputers.” Institute of Electrical and Electronics Engineers (IEEE) Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08).