DRIVING COMPUTING EFFICIENCY

IN ACTUARIAL VALUATION AND ANALYSIS

BY **COREY GRIGG, CHAD SCHUSTER** AND **JIM BRACKET**

Actuaries choosing software platforms must juggle performance targets, cost limits, and rapid delivery schedules. New regulations, frequent re-forecasting, and cloud-usage fees all raise the stakes, requiring coordination with IT, software and cloud vendors, and/or internal development teams. This synopsis, derived from a longer article that will be published in The Actuary, highlights key software and hardware trade-offs to help actuaries with that decision-making process.

WHAT SOFTWARE DEVELOPMENT ISSUES SHOULD YOU CONSIDER?

Programming language dictates both attainable speed and long-term maintainability. Compiled languages (C, C++, C#, Java) deliver raw speed and low-level control but require more programming knowledge to utilize; interpreted or just-in-time-compiled options (Python, Julia, Rust) trade some performance for faster development, automatic memory management, and portability. As an example, the table below shows benchmarking comparison of runtimes in milliseconds for a simple variable annuity simulation implemented in different programming languages:[1] >

COREY GRIGG

[1] For more details, see 'An actuary's guide to Julia: Use cases and performance benchmarking in insurance.'

TABLE 1: BENCHMARKING COMPARISON OF RUNTIMES IN MILLISECONDS FOR A SIMPLE VARIABLE ANNUITY SIMULATION IMPLEMENTED IN DIFFERENT PROGRAMMING LANGUAGES

WINDOWS	C#	C++	JULIA	PYTHON	RUST
Min	4,405	1,754	1,838	30,661	14,153
Mean	4,491	1,775	1,910	45,637	14,301
Max	4,565	1,800	2,132	88,259	14,425

Regardless of the language, code refactoring and optimization can increase performance dramatically, with actuaries adding efficiencies an engineer alone may not realize. For example, using identities from actuarial and financial mathematics, such as put-call parity, or caching of yield curve basis transformations can lead to more efficient code. Actuaries who understand the interdependence of model calculations can help design more efficient processing flows that take advantage of reusable data and parallelism.

WHAT HARDWARE FACTORS SHOULD YOU CONSIDER?

Actuarial workloads vary: some are data-heavy, others calculation-heavy, and many are both. Optimal hardware differs accordingly.

For data-heavy tasks, memory, storage, and networking capabilities must be considered. Ample, low-latency memory can keep full datasets in-process; high-bandwidth SSDs and locality-aware file layouts avert I/O stalls; and high-bandwidth and fast network connections minimize transfer time when source data or results must make their way to the processing machine.

For calculation-heavy tasks, the CPU's clock speed, cache hierarchy, and microarchitecture are paramount. Higher clock speeds achieved through dynamic boosting can lead to faster computations, but they must be balanced with power and heat considerations. Many recent advances in high-performance processing hardware have been focused on reducing memory access latency, particularly by increasing cache sizes and optimizing their design, alongside other architectural enhancements. This minimizes delays from fetching data from slower main memory, which is crucial in computation-heavy scenarios. Modern microarchitectures are designed to optimize parallelism, execute complex instructions more effectively, and reduce latencies faced during multithreaded operations. However, challenges like cache thrashing—where excessive data swapping between cache and main memory degrades performance—can undermine these benefits.

Mixed or scalable tasks require a balanced high-performance computing (HPC) environment. HPC nodes blend large amounts of RAM, many cores, fast SSDs, and high-speed interconnects (e.g., InfiniBand) to support fast calculations, data I/O, and scalability. Single order-of-magnitude performance improvements of actuarial models simply from switching from commodity to HPC hardware are possible.

For calculation tasks that can be massively parallelized, graphics processing units (GPUs) can also prove useful. A GPU is designed for processing multiple similar calculations >

simultaneously, and to some extent can be thought of as a specialized computer with its own RAM, thousands of processors, onboard cooling, and power supply. A GPU relies on a CPU to orchestrate its calculations and to transfer data between the system memory and GPU memory. Thus, a GPU is subject to the constraints of both CPUs (clock speed, memory access patterns, heat/power management) and storage devices (bandwidth, capacity, latency, and input/output access patterns). Some of these constraints can be mitigated through specialized hardware techniques; for example, DMA (Direct Memory Access) or RDMA (Remote Direct Memory Access) can be exploited with CUDA to reduce transfer latency by bypassing CPU participation. Nevertheless, the dependency on a host device can present significant operational and software design challenges. Accordingly, some modeling tasks such as simulation of simple equity-linked products can benefit from GPU if they involve parallelizable calculations driven by simple arithmetic, but for others with features that lead to interdependency of calculations (with-profits products or other asset-liability linkages), large memory requirements (complex policy state tracking for long-term care or management of stateful random number generators), or branching logic (trigger-based policy features, transition-based behavioral models) GPUs may not deliver meaningful performance improvements, or may require modifications to actuarial model logic.

HOW DO YOU OPTIMIZE HARDWARE AND SOFTWARE?

Software that leverages hardware-specific optimizations such as parallelism, vector instructions (SSE/AVX), and GPU off-loading can shrink runtimes dramatically, and modern compilers can often automate much of this. Hardware selection can drive language selection and vice versa. For example, GPU access is simplest in higher-level ecosystems (e.g., Python with Numba) but offers finer tuning in CUDA or OpenCL with C/C++. Hardware-specific tuning boosts speed but reduces portability; thus, payback horizons can be evaluated to justify an investment. For quick wins, consider adjusting firmware via UEFI settings (hyper-threading, memory interleaving, power profiles) or experiment with pre-designed hardware profiles provided by cloud infrastructure vendors to identify out-of-the-box improvements aligned to each model's mix of data and compute.

CONCLUSION

Aligning language choice, hardware profile, and actuarial insight is now essential for delivering faster, more economical valuation and risk analytics. Close collaboration among actuaries, IT, and vendors ensures that each technology layer—code, firmware, and infrastructure—is tuned to workload, sustaining a competitive edge as computational paradigms evolve. <

Chad Schuster

Jim Brackett

COREY GRIGG is Principal & Consulting Actuary, Milliman Chicago. This article has been written with the help of **CHAD SCHUSTER,** Principal Financial Risk Management, Milliman Chicago, and **JIM BRACKETT**, Head of Financial Technology, Milliman Chicago.