High Performance Computing in Finance

presented to Princeton Quant Chicago 2015

John Dodson[1]

OCC & Minnesota Center for Financial and Actuarial Mathematics

November 7, 2015

[1]with acknowledgements to John Reppy and William Gropp.

Computational Aspects

I want to discuss a specific aspect of computation with you today, so let's start by setting a context.

Categories

Computational applications in banking / insurance / investments / trading may be categorized (per ACM) as

- Human Interface for interactive analytics
- Embedded Systems for automated trading
- Data Management for transaction processing
- Communication Design for messaging & workflow
- Knowledge Discovery for data mining
- Computational Modeling for simulation & optimization

While advances in technology and design have revolutionized all of these aspects, I will focus on computational modeling and its application in risk and investment management.

Computational Modeling

Computational modeling is somewhat exotic. Everyone here can probably relate to the technology behind –

- spreadsheets (human interface),
- robots (embedded systems),
- databases (data management),
- the Web (communication design), and even
- IBM's Watson and "big data" (knowledge discovery);

but the poster child for computational modeling is –

- the atom bomb!

Other applications for computational modeling you may know include protein folding, aircraft design, and weather & climate forecasting.

Hardware Concepts

Central Processing Units

Moore's Law, about the rate of technical progression in chip fabrication, held from about 1975 to 2012 and this pattern led to the current dominance of CPU-based architectures.

- CPU-based architectures now support giga-scale computing (10^9 floating-point operations per second), but will probably not advance much beyond that because of quantum and thermo limits.

Parallel Processing

Obscured by the success of CPUs, the technology behind massively-parallel architectures has also progressed. All of the most powerful computers in the world today are based on many-thread / many-data architectures, which are expected to reach the exa-scale (10^{18} FLOPS) later this decade.

Software Concepts

Turing–von Neumann Model

Concepts such as co-mingled code and data, sequential execution, and uniformly-accessible memory led to the development of languages such as Fortran (1957), ALGOL (1958), and its derivatives (C in 1972, C++ in 1983, and Java in 1995). These are the languages we (or at least I!) grew up with. They are designed for CPUs.

- Java will still be running our phones in twenty years ☺

Iverson's APL

An entirely different paradigm emerged in 1964 –

- no loops
- no control structures
- infinite data structures (e.g. $x \triangleq (1, 2, 3, \ldots)$)
- functionals (e.g. $g(f) \triangleq f(\cdot + 1) - f(\cdot)$)

Software Concepts

Data Parallelism

APL was created for mathematicians. It is essentially a dead language today; but by obligating the programmer to think in terms of arrays and easily distributable transformations, the paradigm is finding new life.

Localized Memory

Reading to and writing from common memory is expensive and blocking. That's why we have buses and caches. In the future rather than bringing data to processors, processors will live amongst the data. Everything will always be cached.

Intermediate Libraries

In the meantime, we have work to do!

- MPI: the supercomputer standard from the 1990's
- CUDA: general purpose computing on GPUs since 2007
- OpenCL: an open-source API for heterogenous hardware

Floats

Prior to 1985, there was no standard for representing real numbers in computers. Researchers were doing computational modeling in highly customized settings, sometimes shrouded in secrecy. The industry came together with the IEEE-754 standard which defined representations for floats and doubles.

- The standard was refined somewhat in 2008 with the introduction of decimal128, specifically designed for the needs of financial modeling. Use it!

Other developments in numerics included –

- High-quality pseudo-random generators like the Mersenne Twister in 1997.
- Discontinuous Galerkin, Krylov subspace, and meshfree methods for PDEs developed in the 1970's.

Applications

Risk Measurement

Let's consider the computational complexity involved in measuring the value-of-risk of a portfolio of listed options.

- There are $\sim 10^6$ listed options contracts in the U.S.
- To value each requires a binomial tree with $\sim 10^4$ nodes.
- To get reasonable resolution you need $\sim 10^4$ scenarios.

At the giga-scale, that means a cycle time of $\sim 10^5$ seconds, or about a day. Say you want to –

- perform this calculation intra-day, or
- adjust for estimation risk by re-sampling

then you are in the HPC scale.

Applications

Asset Allocation

Let's consider an example from portfolio optimization. Say you are using the Rockafellar-Uryasev objective, with minimax regret for robustness,

$$\max_{q,\alpha} \min_{\theta \in \Theta} \left(q - \frac{1}{1-c} \mathsf{E}^{\mathbb{P}(\theta)} (q - \Psi_\alpha)^+ \right)$$

with 100 possible assets ($\dim \alpha = 100$) and $\sim$100 possible parameter values for each ($\operatorname{card} \Theta \sim 10^4$). You probably need a Monte Carlo of order $\sim 10^4$ to calculate the expected value for each trial of $q + \alpha$, so now we're up to $\sim 10^8$ calculations per iteration to find an optimum in a 101-dimensional space...

Anyway, you get the point.