nag®

CVA at Scale with Adjoint Sensitivities

Combining the NAG Library with dco/c++ and Origami

Justin Ware, Jacques du Toit, Viktor Mosenkis and Ian Hotchkiss

Xi-FINTIQ

Performance Challenge of XVA

Calculating XVA in a timely manner poses a big performance challenge for financial institutions. The number of trades encompasses the entire institution, not just an individual desk's book(s). It is needed intraday as positions and markets update.

Tools

Practitioners use many tools to achieve the performance required to deliver XVA on time. Some of the most popular are:

Numerical Software Financial modelling typically uses sophisticated numerical software. Here quality (through rigorous testing) and performance are key.

Algorithmic Differentiation (AD) The sensitivities of XVA risk measures to market instruments are often required when reporting results. Estimating these numerically can be very expensive. Algorithmic differentiation, in particular adjoint algorithmic differentiation, offers a more accurate and computationally cheaper alternative.

Grid / Cloud XVA calculations have stages that are highly data parallel *and* computationally intensive. These can be split into multiple tasks which are run on in-house grids and/or public clouds.

The Importance of DAGs

Directed Acyclic Graphs (DAGs) have proven extremely useful in HPC

- They are data-centric: data dependencies are explicit, meaning data movement can be aggressively optimized.
- They describe all levels of parallelism *implicitly*: programmers don't have to try and express it, which can be challenging.
- They are platform agnostic: DAGs can be run on heterogeneous systems with local and remote resources, spanning CPUs and GPUs.
- They are modular: programmers focus on describing individual tasks. Interdependence between tasks describes the overall application.

The figure below shows a DAG for a large-scale adjoint CVA calculation.

Origami: a DAG Execution Engine

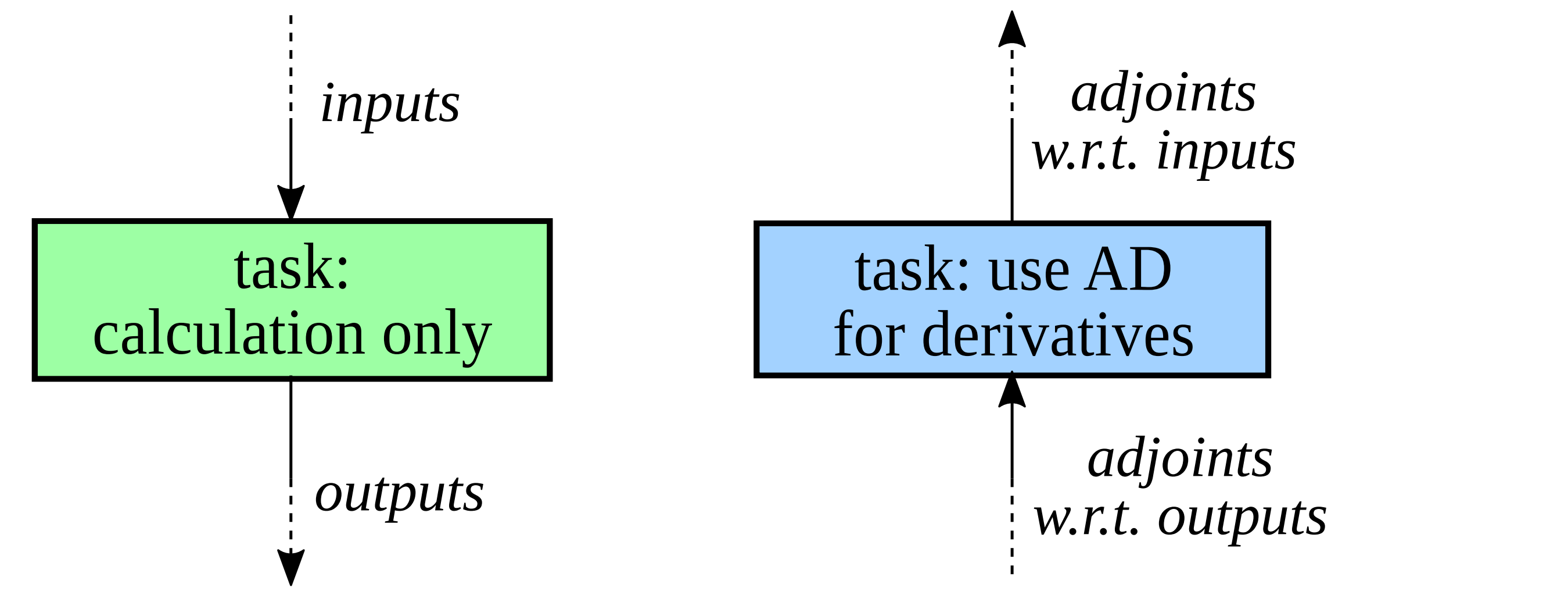
Origami is a light-weight task execution framework which is easy to use and maintain. Users combine tasks into a DAG. Origami can execute the DAG on an ad hoc cluster of workstations on the local network, on a dedicated in-house grid, on production cloud, or on a hybrid of all these. Origami handles all data transfers. Machines can have different hardware, including some with GPUs. Origami reports various statistics including **full accounting figures** for hardware use on a per-task basis. The result is a flexible, high performance execution engine aimed at **business productivity** and **transparency**.

CVA Demonstrator

NAG has developed a CVA demonstration code to show how our (adjoint) numerical libraries and our AD tool `dco/c++` can be combined with Origami to solve large scale CVA computations. Model calibration to market data is in separate tasks. Trades in netting sets are valued in separate batches. CVA is calculated per netting set, and sensitivities with respect to market instruments are computed using adjoint AD. The resulting DAG has a large number of tasks with non-trivial dependencies (see below) which Origami can automatically process and execute.

Combining Tools

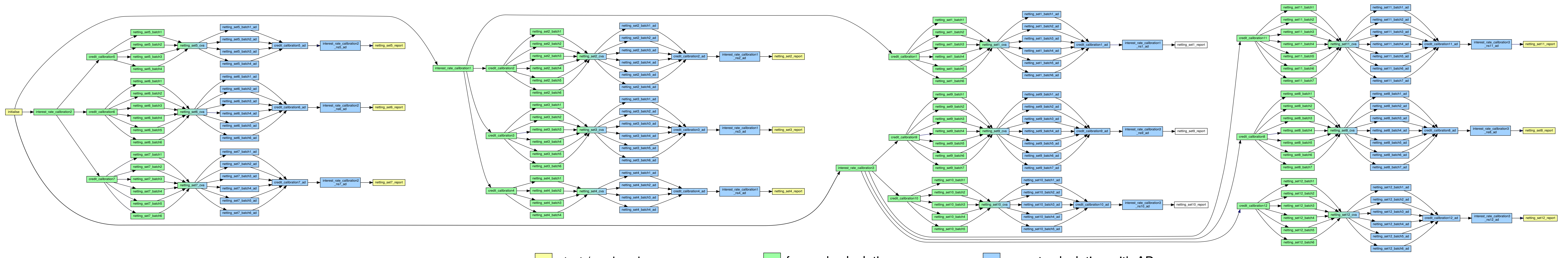
- `dco/c++` is an (A)AD tool computing sensitivities of C++ codes via operator overloading. Since it operates at the source code level, closed source 3rd party dependencies (such as the NAG Library) cause problems.
- AD-enabled (adjoint and tangent) versions of NAG Library routines are available which interface seamlessly with `dco/c++`. These allow adjoint calculations to pass through NAG Library dependencies.
- Creating an adjoint of a distributed code is straightforward.
- The entire application is run forward as normal.
- The DAG is then reversed and the `dco/c++` adjoint version of each task is run.



- The adjoint propagates sensitivities backward through the task. Adjoints are taken seamlessly through NAG Library routines.
- Origami handles the data movement of adjoints between tasks.

Availability

These tools are available to trial. Please contact **infodesk@nag.co.uk** to arrange access.



🟨 start / end nodes 🟩 forward calculation 🟦 repeat calculation with AD