High Performance Tape–Free Adjoint AD for C++11

Introducing dco/map, a cross–platform, accelerator ready AAD tool

Jacques du Toit (NAG), Johannes Lotz (RWTH), Klaus Leppkes and Uwe Naumann

Making Adjoints of C++ Codes

The C++ language is so complex that no AD compilers can handle it. To get an adjoint, we must write it by hand or use an operator overloading AAD tool. Handwritten adjoints are tedious to write and maintain, while operator overloading tools have a tape which must be managed. The tape is particularly problematic on accelerators since each thread (or potentially each AVX lane) needs its own. On GPUs this is just impractical.

Meta Adjoint Programming

dco/map is a tape–free operator overloading AAD tool. Since it uses operator overloading it can handle many C++11 codes. It uses meta programming to instantiate sections of the adjoint code at compile time, re–using program data on the stack. This technique is called meta adjoint programming and creates highly efficient adjoint codes.

Key Features of dco/map

- Efficient adjoints through C++11 meta adjoint programming
- First and second order tangents
- First order adjoints (second order in development)
- Thread safe by design: high performance array and scalar types for shared input data
- Unified code: a single source code can be run passively, in tangent mode or in adjoint mode
- Support for user defined custom “tape”
- Mixed precision adjoint types in development
- Supports CUDA and OpenMP (AVX512 SIMD in development)
- Supports Windows (VS2013 and higher) and Linux (gcc 4.6.0 and higher)
- Existing source will need refactoring to fit with dco/map programming model

The tool is already being used in pre-production XVA projects.

An In–House CVA Prototype

NAG has produced an in–house prototype CVA code demonstrating how to combine dco/map and CUDA to accelerate CVA applications. The code uses the G2++ model with wrong way risk and the netting set is a portfolio of swaps. Setup, calibration and final aggregation is on the CPU using tape based AD, while the Monte Carlo uses dco/map and can be run on CPU or GPU with minor code changes.

Code Samples from the CVA Prototype

```
template<class Base> class CurveInterpolator {
   /* ... snip ... */
   template<class Active> class Functions {
      const dco_map::array_t<Active,dco_map::reduction_push> knots;
      const dco_map::array_t<Active,dco_map::reduction_push> curveValues;
      // Simple linear interpolation
      FUNDECL void valueAt(const Base & t, Active &out) const {
         int i;
         for(i=0; i<n; i++) { if( knots[i]>=t) { break; } }
         out =  curveValues[i] +  (curveValues[i+1] - curveValues[i])
                              / ( knots[i+1] - knots[i] ) * (t-knots[i]);
      }
      /* ... snip ... */
};

template<class Base> class G2InterestRateModel {
   /* ... snip ... */
   template<class Active> class Functions {
      const dco_map::connector_t<Active> a, b, rho, x0, y0;
      const CurveInterpolator<Base>::Functions<Active> sigmaCurve,
         etaCurve, timeZeroRatesCurve, timeZeroForwardSpread;

      // Bond price in the G2++ model: V() is a class member function
      FUNDECL void zeroCouponBondPrice(const Base &t, const Base &T,
                     const Active &xt, const Active &yt, Active &ret) {
        Active rT;
        MAP_CALL(Active, timeZeroRatesCurve.valueAt(T, rT) );
        const auto zeroBond_0_T = exp(-rT * T);
         MAP_IF(Active, t <= 0) {
            ret = zeroBond_0_T;
         } MAP_ELSE {
            Active sigmat, etat, rt, v;
            MAP_CALL(Active, timeZeroRatesCurve.valueAt(t, rt) );
            const auto zeroBond_0_t = exp(-rt * t);
            MAP_CALL(Active, sigmaCurve.valueAt(t, sigmat) );
            MAP_CALL(Active, etaCurve.valueAt(t, etat) );
            MAP_CALL(Active, V(t, T, sigmat, etat, v) );
            ret = zeroBond_0_T/zeroBond_0_t * exp( 0.5*v -
                  (1-exp(-a*(T-t)))*xt/a  - (1-exp(-b*(T-t)))*yt/b );
         } MAP_IF_END;
      }
      /* ... snip ... */
};

template<class Base, class Active> __global__ void makeSamplePaths( ... )
{
   const G2InterestRateModel<Base>::Functions<Active> g2model(...);
   dco_map::array_t<Active> xt(nPaths*nEulerSteps,d_xt, d_a1_xt);
   dco_map::array_t<Active> yt(nPaths*nEulerSteps,d_yt, d_a1_yt);

   const auto rho = g2model.rho; const auto corr = sqrt(1-rho*rho);
   for(int path=tid; path<nPaths; path += blockDim.x*gridDim.x) {
      MAP_FOR(Active, time, 0, nEulerSteps-1, 1) {
         // Read the random numbers and correlate them
         const auto z1 = Z(path,time,0);
         const auto z2 = Z(path,time,1);
         const auto dW1 = z1; const auto dW2 = rho*z1 + corr*z2;
         const t = time*dt;
         Active xi, yi, xii, yii;
         MAP_IF(Active, time==0) {
            xi = g2model.x0;          yi = g2model.y0;
         } MAP_ELSE {
            xi = Xt(path,time-1);     yi = Yt(path,time-1);
         } MAP_ENDIF;
         MAP_CALL(Active, g2model.applyEulerIncrement(time, dt, dW1, dW2,
                                                      xi, yi, xii, yii) );
         Xt(path,time) = xii;
         Yt(path,time) = yii;
      }
      MAP_FOR_END;
   }
}
/* ... snip ... */
```

Results for CVA Prototype

The table below gives runtimes (in ms) and memory use (in MB) for the whole (heterogeneous) application:

	gcc	clang	VS2015	nvcc : total	nvcc : GPU	Mem
passive	1,650	2,070	900	184	15	68
dco/map	5,024	5,370	24,590	370	111	176
tape	12,440	9,310	23,500	N/A	N/A	4,890

Results for Local Volatility Monte Carlo Kernel

The table below gives runtimes (in ms) and memory use (in MB) of a local volatility FX basket pricer. There are 10 assets and the basket is priced with Monte Carlo using 10,000 sample paths and 360 time steps. The tape based implementation features an optimised checkpointing strategy and uses virtually the same amount of memory as a handwritten adjoint. Nevertheless the dco/map runtime is substantially less due to the meta adjoint programming

Linux				
	clang	gcc	nvcc	Mem
passive	1,461	1,406	18	430
dco/map	3,031	3,025	83	827
tape	13,579	11,011	N/A	835
Windows				
	clang	VS2015	Intel2015	Mem
passive	1,172	1,510	1,421	430
dco/map	4,384	10,241	11,671	827
tape	16,125	24,025	18,833	835

To arrange access to dco/map please email **support@nag.co.uk**