Algorithmic Differentiation: Handwritten vs Runtime

- Handwritten adjoint: most efficient, but laborious to write, error-prone and have two sets of source to be kept in sync.
- Runtime: primal code executed through a tool which builds execution graph (tape) at runtime and computes adjoint from this. Very flexible, only one source, but has runtime overheads and potentially massive memory use.

Runtime AD tools such as dco/c++ are most popular due to flexibility and productivity. Memory use of dco/c++ can be constrained as much as desired, and runtime overheads are small.

Runtime AD Tools are not Always Appropriate

Sometimes handwritten adjoints are preferable, or are the only option:
- Accelerators such as GPUs: thousands of threads, limited RAM and branching control logic makes taping hard or infeasible
- “Static” library routines (e.g. spline evaluation) which don’t change and are heavily used
- Performance-critical kernels, e.g. Monte Carlo
- Applications where available memory is constrained

Compile-time Adjoint Thes through Operator Overloading

With C++11 it is possible to write a meta-program based overloading tool which uses the C++11 compiler (gcc, cl, nvcc, etc.) to instantiate an adjoint code at compile time. If the tool is designed carefully, and the platform compiler is good enough, then the resulting code can be optimized to the same level as a handwritten adjoint. We implemented this in a new dco/c++ type called dco_ntr.

Multi-dimensional Local Volatility Test Code

We applied the tool to an FX basket code: 10 factor local volatility model driving a basket of 10 FX rates, local volatility surfaces represented by 1D splines. The model has about 450 inputs, mainly the observed market implied volatility quotes. Price and full gradient are computed on GPU or CPU (serial only).

Adjacent code snippets show the Monte Carlo kernel. Minor details are omitted for brevity. The function compute is called from CPU code or GPU kernels and instantiated with dco_ntr scalar and array types or normal (float, float*) types. It stores Monte Carlo paths in d_logs and checkpoints local volatilities in d_locVolCKP which are used to speed up the adjoint run.

Local Volatility Monte Carlo Source Code

```cpp
template<class FP, class ARRAY>
void getVol(const FP S0, int n, const ARRAY& array, const ARRAY& coeff, const ARRAY& extrap, FP sigmas) {
    // Linearly extrapolate in the wings
    if(x <= knots[3]) {
        sigmas = extrap[0] + extrap[1]*x; // x < knots[3];
    } else if(x >= knots[0]) {
        sigmas = extrap[2] + extrap[3]*x - knots[0];
    } else {
        sigmas = evalSpline(x, knots, coeff);
    }
}
```

Results for Local Volatility Monte Carlo Kernel

The table below gives runtimes (in ms) of the Monte Carlo kernel (passive), the handwritten adjoint (handwritten), the adjoint via the new dco_ntr type, and the adjoint via dco/c++ tape. Figures in brackets are adjoint time as multiple of passive time.

<table>
<thead>
<tr>
<th></th>
<th>clang 3.6</th>
<th>gcc 4.7</th>
<th>nvcc 7.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>passive handwritten</td>
<td>1,461</td>
<td>1,406</td>
<td>18</td>
</tr>
<tr>
<td>dco_ntr</td>
<td>1,906</td>
<td>2,808</td>
<td>89 (4.9x)</td>
</tr>
<tr>
<td>dco/c++ tape</td>
<td>3,031</td>
<td>3,025</td>
<td>83 (4.6x)</td>
</tr>
<tr>
<td>N/A</td>
<td>13,579</td>
<td>11,011</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Results are all on the same machine equipped with a K20c graphics card. Note that CUDA 7.5 does not support C++11 on Windows. The clang compiler is truly impressive on both platforms. Intel 15.0 on Linux gives dco_ntr runtime of 9,725ms. Although results on Windows are not as good as on Linux, the memory use on all platforms is always as efficient as a handwritten adjoint.

Benefits of New dco/c++ Type

- Single “natural looking” source for both primal and adjoint
- Primal code can be changed and adjoint is always in sync
- Adjoint runtime often as fast as handwritten, and much faster than tape
- Memory use as efficient as handwritten adjoint
- The dco_ntr type works with any C++11 compiler

The new type system is under development and will be part of a future dco/c++ release. NAG encourages anyone interested in the technology to contact us as supported early access can be arranged and would be beneficial to both parties.

Acknowledgments

NAG would like to thank Prof. Dr. David Bommes of RWTH Aachen for making code available during early testing.