Black Scholes 2014

Here is the Black Scholes equation from wikipedia:

\[
C(S, t) = N(d_1)S - N(d_2)Ke^{-r(T-t)}
\]

\[
d_1 = \frac{1}{\sigma \sqrt{T-t}} \left[ \ln \left( \frac{S}{K} \right) + \left( r + \frac{\sigma^2}{2} \right)(T-t) \right]
\]

\[
d_2 = \frac{1}{\sigma \sqrt{T-t}} \left[ \ln \left( \frac{S}{K} \right) + \left( r - \frac{\sigma^2}{2} \right)(T-t) \right]
\]

\[
d_1 = d_2 - \sigma \sqrt{T-t}
\]

\[
N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}z^2} \, dz
\]

- \( N() \) is the cumulative distribution function of the standard normal distribution
- \( T - t \) is the time to maturity
- \( S \) is the spot price of the underlying asset
- \( K \) is the strike price
- \( r \) is the risk free rate (annual rate, expressed in terms of continuous compounding)
- \( \sigma \) is the volatility of returns of the underlying asset

What is competitive performance for Black Scholes valuation in late 2013? The most straightforward way to get a competitive execution time estimate is to count the required multiplies and additions. We could do that directly from the equations, making some assumptions about common subexpression elimination and caching. But we can simply analyze Intel’s code for Black Scholes (below), that’s easier for now. Once we know the number of multiplies and additions we can estimate how many execution cycles we will need on a given microprocessor executing this code. Since we can look up the clock speed of the given microprocessor we can back into an estimate of the time per Black Scholes valuation. The last time we did this analysis in 2009, here, we found that on a 2007 vintage microprocessor (IBM POWER 6) that we needed 170 cycles for the
valuations and input sensitivities. That was about 36 nanoseconds per valuation or about 30 million Black Scholes valuations per second. What has changed in competitive Black Scholes performance in the past five years?

Here is Intel code implementing the Black Scholes equation valuation.

```c
void BlackScholesFormula( int nopt, tfloat r, tfloat sig, tfloat s0[], tfloat x[], tfloat t[], tfloat vcall[], tfloat vput[] )
{
    vmlSetMode( VML_EP );
    DIV(s0, x, Div); LOG(Div, Log);
    for ( j = 0; j < nopt; j++ ) { // loop 1
        tr[j] = t[j] * r;
        tss[j] = t[j] * sig_2;
        tss05[j] = tss[j] * HALF;
        mtr[j] = -tr[j];
    }
    EXP(mtr, Exp); INVSQR(tss, InvSqrt);
    for ( j = 0; j < nopt; j++ ) { // loop 2
        w1[j] = (Log[j] + tr[j] + tss05[j]) * InvSqrt[j] * INV_SQRT2;
        w2[j] = (Log[j] + tr[j] - tss05[j]) * InvSqrt[j] * INV_SQRT2;
    }
    ERF(w1, w1); ERF(w2, w2);
    for ( j = 0; j < nopt; j++ ) { // loop 3
        w1[j] = HALF + HALF * w1[j];
        w2[j] = HALF + HALF * w2[j];
        vcall[j] = s0[j] * w1[j] - x[j] * Exp[j] * w2[j];
        vput[j] = vcall[j] - s0[j] + x[j] * Exp[j];
    }
}
```

Let’s estimate the theoretical competitive performance of this code. We will assume double precision although single precision would be significantly faster. We will assume LA rather than EP for VML execution. We will add some cycles to the estimates coming from this code to account for input sensitivities. That will make these estimates directly
comparable to the 2009 estimates. We need some current cycles counts from Intel VML, here. We use the counts labeled Intel® Core™ i5-4670T Processor base clocked at 2.3 GHz turbo to 3.3 GHz.

<table>
<thead>
<tr>
<th>Code</th>
<th>Cycles/Element</th>
<th>Accuracy ULP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Div()</td>
<td>3.02</td>
<td>3.10</td>
</tr>
<tr>
<td>Log()</td>
<td>6.16</td>
<td>0.80</td>
</tr>
<tr>
<td>loop 1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Exp()</td>
<td>3.65</td>
<td>1.98</td>
</tr>
<tr>
<td>Invsqrt()</td>
<td>3.70</td>
<td>1.42</td>
</tr>
<tr>
<td>loop2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>ERF() x 2</td>
<td>6.05*2</td>
<td>1.33</td>
</tr>
<tr>
<td>loop3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>37.63</td>
<td></td>
</tr>
</tbody>
</table>

The sensitivities drop out from differentiating the Black Scholes formula. We have lots of common subexpressions:

\[
\begin{align*}
\delta &= N(d_1), \\
\gamma &= N(d_1)/(S\sigma t), \\
\vega &= S N(d_1) \sqrt{t}, \\
\theta &= -(S N(d_1) \sigma)/2 \sqrt{t} - r K \exp(-r t) N(d_2), \text{ and} \\
\rho &= K t \exp(-r t) N(d_2)
\end{align*}
\]

This looks like 6 or maybe 7 cycles of computation. I don’t see a way to argue away the 3 cycles for the divide in the gamma. Let’s say 44 cycles all in for this estimate on a single core. So, a full Black Scholes valuation is executed every 13 nanoseconds @ 3.3 GHz and about 19 nanoseconds @ 2.3 GHz if there is no L1/L2 cache pressure on the core. This code is not going to generate many cache misses.

The i5-4670T has four cores let’s assume you can use three of them trivially @2.3 GHz and the OS uses the 4th. You can get to 150 million full Black Scholes valuations per second on one i5-4670T, if you can somehow get to execute on that 4th core you could crack 200 million per second. It does not look like Turbo boost is going to get you 4 cores running at 3.3 GHz. Maybe over-clocking the 4670 will let you crack 200 million full Black Scholes valuations per second. I’d estimate competitive performance in late 2013 is 150 to 200 million full Black Scholes per second on a single i5-4670T at $213 a pop, up from 30 million per second in 2009.