Aileen Foreman

Mathematical Algorithms to Maximize Performance in Numerical Weather Prediction

Introduction

Numerical weather prediction models, which involve the solution of non-linear partial differential equations at points on an extensive three-dimensional grid, are ideally suited for processing on vector machines. It was logical therefore that the new global forecast model to be implemented at the Meteorological Office should be written in vector code for the Cyber 205.

In order to achive full efficiency and to reduce storage requirements the model used 32-bit arithmetic which had been found to provide high enough precision. Unfortunately, however, the trigonometrical and logarithmic functions provided by CDC could only handle 64-bit vectors and, although written in efficient scalar code, did not take advantage of the special facilities of a vector processor. It was therefore necessary to rewrite the functions in vector code to handle both 32 and 64-bit vectors. There was also no half-precision compiler available for the Cyber 205 at that time and so the functions, like the model, had to make extensive use of the "special call" syntax. This made the code more difficult to write but it allowed much greater flexibility in that it became possible to access the exponent of a floating-point number independently of its coefficient.

This paper presents a description of the techniques and it summarises the results which were achieved. One example, the logarithmic function, is treated here in detail to illustrate the general approach to the problem.

Derivation of logarithms

The coding for the logarithm function illustrates both the use of the way in which floating-point numbers are stored and the use of linked triads to gain additional speed.

To calculate $y = \log_e(x)$ we divide the range of x into two, the first of which is

a) $x \geq \sqrt{2}$ and $x < \frac{\sqrt{2}}{2}$

We first write the value of x in a way which can be related to the format of stored floating-point numbers. Thus, introducing two new unknowns n and ω, n being an integer and $\frac{1}{2} \leq \omega \leq 1$, we may write any number as $x = 2^n \omega$.

Now the Cyber 205 stores the floating-point number as

$$2^{exp} \cdot coefficient = 2^{exp} \cdot 2^{j} \cdot k$$

where the factor 2^j is introduced by normalization.

Since for logarithms, x must always be positive, for 64-bit numbers bit 17 will be on, so j = 46 and for 32-bit numbers bit 9 will be on, so j = 23.

Then relating the two, we have $n = exp + j$ and $\omega = k$

As an example, if x = 2.0 as a 64-bit normalized value

$$x = 2^{-45}.2^{46}$$

so from the above formulae

$$n = -45 + 46 = 1 \qquad \text{and} \qquad \omega = 1.0$$

Here, we can obtain the values of n and ω very easily as we can access the exponent and coefficient of a number by using special calls.

The next step is to convert the functions into a suitable form for vectorization and this involves the introduction of a new variable

$$z = \left(\frac{\omega - \sqrt{2}/2}{\omega + \sqrt{2}/2}\right)$$ which can be computed at the same time as ω .

$$\text{Then} \quad \omega = \left(\frac{1+z}{1-z}\right)\frac{\sqrt{2}}{2}$$

From the original definition

$$x = 2^{n-1/2}\left(\frac{1+z}{1-z}\right)$$

thus $$\log_e x = \left(n - \frac{1}{2}\right)\log_e 2 + \log_e\left(\frac{1+z}{1-z}\right)$$

b) For the remaining values of x , within the range $\frac{\sqrt{2}}{2} \leq x < \sqrt{2}$, the value of z is defined by:

$$z = \frac{x-1}{x+1} \qquad \text{so that} \qquad x = \frac{1+z}{1-z}$$

Then $$\log_e x = \log_e \frac{1-z}{1+z} \qquad \text{for } \frac{\sqrt{2}}{2} \leq x < \sqrt{2}.$$

In each case, the problem then becomes one of vectorizing $\log_e\left(\frac{1+z}{1-z}\right)$ which is easily done by replacing it with a truncated series which gives the required degree of precision:

$$\log_e\left(\frac{1+z}{1-z}\right) = \sum_{m=0}^{6} c_m z^{2m+1}$$

where the constants c_m are known.

Then $\log_e\left(\frac{1+z}{1-z}\right) =$

$$((((((c_6 z^2 + c_5) z^2 + c_4) z^2 + c_3) z^2 + c_2) z^2 + c_1) z^2 + c_0) z$$

Despite its complicated appearance, this reduces to eight vector operations consisting of a multiplication, six linked triads and a final multiplication by z thus

Multiplication to give z^2

First triad = V1 = $c_6 z^2 + c_5$

Second triad = V2 = $V1\, z^2 + c_4$

Third triad = V3 = $V2\, z^2 + c_3$ etc.

Tests, using the 1.5 compiler, and a range of vector lengths gave the following results, with times being expressed in units of 10^{-4} seconds.

Vector length	10	50	100	200	500	1000	2000	5000
CDC logarithms	.3	.55	.7	1.01	2.00	3.66	7.04	21.50
64-bit vector logarithm	.47	.61	.78	1.12	2.16	3.87	7.47	20.15
32-bit vector logarithm	.53	.57	.65	.82	1.34	2.20	3.99	9.66

The first point to notice here is that the full increase in speed for 32-bit vectors is only achieved with large vector lengths. Because of the overheads associated with the initiation of vector instructions, this is not unexpected and is common to all of the functions to be described. What is unexpected is that no improvement in speed was achieved for our 64-bit function when compared to the CDC function. In this respect, this function is unique among all those treated in this paper. However, the original aim of producing a 32-bit version has been successfully achieved.

Exponentials

The exponential function is derived from the standard formula

$e^x = 2^k . 2^{m/16} . 2^{f/16}$ chosen to make use of special calls. k, m and f are defined as follows:

If $n = int\left[\frac{16x}{\log_e 2}\right]$

then $k = int\left[\frac{n}{16}\right]$ and $m = n$ modulo 16 for $x \geq 0$

and $k = int\left[\frac{n}{16}\right] - 1$ and $m = 16 - n$ modulo 16 for $x < 0$

$$f = \left(\frac{16x}{\log_e 2}\right) - n$$

Now, since m is integer and $0 \leq m < 16$, the factor $2^{m/16}$ is obtained from a look-up table of 16 elements of known values, using the "special call" instruction Q8VXTOV.

Having found the integer k from the above formula, and $2^{m/16}$ from the look-up table, to obtain the value $2^k . 2^{m/16} = 2^{k+m/16}$ we add k to the exponent part of $2^{m/16}$ by using special calls.

The factor, $2^{\delta/16}$ is given by

$$2^{\delta/16} = \frac{P_3\delta^3 + \delta^2 + P_1\delta + P_0}{-(P_3\delta^3 - \delta^2 + P_1\delta - P_0)}$$

where f is obtained as above and P_0, P_1, P_3 are known constants.

Then, to obtain e^x all we need is a final multiply of $2^{\delta/16}$ by $2^{n+m/16}$

The following results were achieved, times again being given in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC exponential	.35	.7	.93	1.44	2.86	5.25	10.52	33.36
64-bit vector exponential	.47	.6	.78	1.14	2.29	4.15	7.97	22.75
32-bit vector exponential	.47	.56	.68	.93	1.85	3.14	5.85	14.62

Here, for a vector length of 5000 the 32-bit exponential routine is only 40% faster than the 64-bit routine because of the use of the "special call" Q8VXTOV. However the 64-bit routine has achieved a considerable speed-up over the CDC exponential.

The Hyperbolic functions

The routines to calculate the hyperbolic functions $y=\cosh x$, $y=\sinh x$ and $y=\tanh x$ use the following formula,

$$\cosh x = \frac{1}{2}\left(e^x + e^{-x}\right)$$

The calculation of e^x is as described earlier. During the calculation of e^x, little extra work is required to obtain e^{-x} which avoids the need to call the exponential routine twice.

The hyperbolic sine is given by

$$\sinh x = \frac{1}{2}\left(e^x - e^{-x}\right) \qquad \text{for} \quad |x| \geq 0.5$$

and

$$\sinh x = \sum_{m=0}^{5} \frac{x^{2m+1}}{(2m+1)!} \qquad \text{for} \quad |x| < 0.5$$

Here the two distinct cases are treated independently, so that we are dealing with shorter vector lengths, and then the results are merged together at the end of the routine. The polynomial expansion of sinh x can be performed in seven vector instructions, by using linked triads.

The hyperbolic tangent is given by

$$\tanh x = \sum_{m=0}^{5} c_m x^{2m+1} \qquad \text{for} \quad 0 \leq |x| \leq 0.12$$

$\tanh x = 1 - \frac{2}{e^{2x}+1}$ for $0.12 < |x| \leq 18.0$

$\tanh x = 1.0$ for $x > 18.0$

$\tanh x = -1.0$ for $x < -18.0$

Again, the distinct cases are treated independently so that we are dealing with shorter vector lengths, and again we can use linked triads when calculating the polynomial expansion of $\tanh x$.

The timings of the hyperbolic sine and hyperbolic tangent routines are data dependent, but some sample timings are given below. All times are expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
hyperbolic cosine								
64-bit vector	.55	.79	1.08	1.68	3.45	6.41	13.26	37.65
32-bit vector	.54	.69	.88	1.27	2.44	4.44	8.72	22.99
hyperbolic sinh								
64-bit vector	.75	.99	1.30	1.96	3.88	7.27	14.87	43.85
32-bit vector	.72	.87	1.07	1.48	2.74	5.00	9.47	24.38
hyperbolic tangent								
64-bit vector	.66	.87	1.15	1.68	3.33	6.01	11.79	34.83
32-bit vector	.64	.73	.89	1.21	2.30	3.66	6.87	17.76

Again, we see that for very short vector lengths we do not have a great advantage by using 32-bit vectors, but for longer vector lengths we are approaching twice the speed of the 64-bit functions. There were no CDC functions available to compare with our results.

Sines and cosines

The trigonometrical functions, $y = \sin x$ and $y = \cos x$ are calculated from the polynomial expansion of $\sin x$ so that we can make use of linked triads again. First the input argument needs to be reduced modulo 2π. This is achieved by

letting $r_1 = \frac{2|x|}{\pi}$ and $r_2 = int\left[\frac{2|x|}{\pi}\right]$

then put $z = r_1 - r_2$ so that $0 \leq z < 1$.

and $k = r_2$ modulo 4

So $\sin(x)$ is given by

$\sin x =$ $\sin z$ for $k=0$

$\sin(1-z)$ for $k=1$

$-\sin z$ for $k=2$

$-\sin(1-z)$ for $k=3$

where $\sin z = \sum_{m=0}^{8} c_m z^{2m+1}$ for 64-bit function

and the constants c_m are known.

Because the values c_7 and c_9 are too small to affect the accuracy of the 32-bit function results:

$$\sin z = \sum_{m=0}^{6} c_m z^{2m+1}$$ for 32-bit vector function

The cosine function is given by

$$\cos x = \sin\left(\frac{\pi}{2} + x\right)$$ where $\sin\left(\frac{\pi}{2} + x\right)$ is calculated as above.

If it is known that the input operand, x, is always between -2π and $+2\pi$ radians, much work can be left out of the routine;

for as above let $r_1 = \frac{2|x|}{\pi}$ and $r_2 = int\left[\frac{2|x|}{\pi}\right]$

and so $0 \le r_2 \le 3$

and $k = r_2$ modulo $4 = r_2$

and again $z = r_1 - r_2$ so $0 \le z < 1$

So for $k = r_2 = 0$, $z = r_1 - r_2 = r_1$, $\sin x = \sin(z) = \sin(r_1)$

for $k = r_2 = 1$, $z = r_1 - r_2 = r_1 - 1$, $\sin x = \sin(1 - z) = \sin(2 - r_1)$

for $k = r_2 = 2$, $z = r_1 - r_2 = r_1 - 2$, $\sin x = -\sin(z) = \sin(2 - r_1)$

for $k = r_2 = 3$, $z = r_1 - r_2 = r_1 - 3$, $\sin x = -\sin(1 - z) = \sin(r_1 - 4)$

Thus we have two sets of functions, one set to calculate the sine and cosine of any angle expressed in radians, and the other to calculate the sine and cosine of angles between -2π and $+2\pi$ radians.

The polynomial expansion of sin(z) can be calculated in ten vector instructions including eight linked triad instructions for the 64-bit function and in eight vector instructions using six linked triad instructions for the 32-bit functions.

Tests gave the following results with times given are expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC sine	.15	.5	.64	.91	1.72	3.07	6.13	22.98
64-bit vector sine (all angles)	.49	.59	.72	.98	1.74	3.02	5.59	14.98
32-bit vector sine (all angles)	.42	.46	.52	.63	.98	1.57	2.76	6.35
64-bit vector sine (-2π to $+2\pi$)	.37	.44	.53	.72	1.27	2.20	4.07	10.04
32-bit vector sine (-2π to $+2\pi$)	.34	.37	.41	.50	.75	1.20	2.09	4.78

vector length	10	50	100	200	500	1000	2000	5000
CDC cosine	.3	.55	.68	.99	2.08	3.29	6.68	23.59
64-bit vector cosine (all angles)	.57	.60	.73	.99	1.87	3.19	5.94	16.00
32-bit vector cosine (all angles)	.69	.47	.51	.63	1.0	1.70	2.94	6.95
64-bit vector cosine (-2π to $+2\pi$)	.72	.45	.55	.74	1.42	2.40	4.45	11.14
32-bit vector cosine (-2π to $+2\pi$)	.67	.37	.41	.50	.77	1.37	2.31	5.51

Thus, we can see that we need a vector length of 500 to 1000 before our 64-bit routines for all angles are faster than the CDC supplied routines, but that our 32-bit routines for restricted angles between -2π and $+2\pi$ are over four times as fast as the CDC routines for vector lengths of 5000.

Tangents

Similarly for the trigonometrical function, $y = \tan x$ we have supplied two sets of functions, one set to calculate the tangent of any angle expressed in radians in both 64-bits and the other to calculate the tangent of angles between -2π and $+2\pi$ radians in both 64-bits and 32-bits. The tangent function is calculated using a polynomial expansion of tan(x) to make use of linked triads. The calculation is performed by first reducing the argument modulo 2π

Let $r_1 = \frac{4x}{\pi}$ and $r_2 = int\left[\left|\frac{4x}{\pi}\right|\right]$

then $z = r_1 - r_2$ so that $0 \le z < 1$

Now let $s = r_2$ modulo 8, putting $k = s$ if $0 \le s \le 3$
and $k = s-4$ if $4 \le s \le 7$

$\tan(x)$ is now given by

$$\tan(x) = \tan(z) \quad \text{for} \quad k=0$$

$$= \frac{-1}{\tan(z-1)} \quad \text{for} \quad k=1$$

$$= \frac{-1}{\tan(z)} \quad \text{for} \quad k=2$$

$$= \tan(z-1) \quad \text{for} \quad k=3$$

where $\tan(z) = \sum_{m=0}^{12} c_m z^{2m+1}$ to the required degree of precision.

Again, if it is known that the input operand is always between -2π and $+2\pi$ radians, we can write:

$$r_1 = \frac{4x}{\pi} \qquad \text{and} \qquad r_2 = \text{int}\left[\left|\frac{4x}{\pi}\right|\right]$$

and so $0 \le r_2 \le 7$

In this case $s = r_2$ modulo $8 = r_2$

Then $k = r_2$ where $0 \le r_2 \le 3$
and $k = r_2 - 4$ where $4 \le r_2 \le 7$

and the calculation continues as before.

The polynomial expansion of tan(z) is calculated in fourteen vector instructions using twelve linked triads.

The resulting timings of tests are given below, expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC tangent	.98	.73	.91	1.47	2.61	4.71	9.33	30.80
64-bit vector tangent (all angles)	.90	.82	.99	1.35	2.55	4.48	8.40	22.67
32-bit vector tangent (all angles)	.96	.78	.90	1.14	1.92	3.21	5.59	13.29
64-bit vector tangent (-2π to $+2\pi$)	.67	.76	.93	1.25	2.36	4.14	7.74	20.64
32-bit vector tangent (-2π to $+2\pi$)	.67	.70	.80	.99	1.76	2.94	5.15	11.98

These results show that we need a vector length of only about 200 before our 64-bit tangent function for all angles is faster than the CDC routine, and that our 32-bit tangent function for restricted angles between -2π and $+2\pi$ radians is well over twice as fast as the CDC routine.

The Arctangent function

The arctangent function $y = a\tan(x)$ is again calculated from a polynomial expansion so that we can use linked triads. The calculation is performed as follows:

For $|x| \geq \sqrt{2}+1$ let $\omega = \frac{1}{|x|}$

and for $|x| < \sqrt{2}+1$ let $\omega = |x|$

Change the variable to z, defined by

$$z = \frac{\omega - a}{a + \omega a^2}$$

where, a is chosen so that z = 1.0 when $\omega = \sqrt{2}+1$

Under this condition, $a = (1-\sqrt{2}) + \sqrt{4 - 2\sqrt{2}}$, and is therefore a constant.

Then atan(x) is given by

atan(x)=atan(z)+atan(a)

Here, atan(a) is a constant and need only be calculated once, and we may replace atan(z) by the truncated series:

$$a\tan(z) = \sum_{m=0}^{4} c_m z^{2m+1}$$

For $|x| \geq \sqrt{2}+1$, $a\tan(x) = \frac{\pi}{2} - a\tan\left(\frac{1}{x}\right)$

and for $x < 0$, $a\tan(x) = -a\tan(x)$

Atan(z) can be calculated in ten vector instructions, eight of which are linked triad instructions. The results are in the range $-\frac{\pi}{2}$ to $+\frac{\pi}{2}$ (not inclusive).

The following results were achieved, times again being given in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC arctangent	.38	.90	1.19	1.97	4.06	7.35	16.19	46.09
64-bit vector arctangent	.48	.52	.66	.92	1.91	3.07	5.77	15.23
32-bit vector arctangent	.43	.49	.55	.69	1.10	1.79	3.34	7.27

These results are spectacular, in that the 32-bit arctangent function is over six times as fast as the CDC routine and even the 64-bit version has given a threefold increase in speed.

Derivation of arcsine and arccosine functions

The final trigometric routines to be considered calculate the arcsine and arccosine of x. The calculations are performed as follows.

for $0 \le x \le 1/2$, let $z = x$ so that asin(x) = asin(z)

and for $\frac{1}{2} < x \le 1$, let $z = \left(1 - \frac{x}{2}\right)^{1/2}$ and asin(x) = $\frac{\pi}{2} - 2\,asin(z)$

for $-1 \le x < 0$, asin(x) = asin(-x) and the same substitutions are used.

Now the new variable, z, must be between zero and 0.7 so we may write

$$asin(z) = \sum_{n=0}^{11} C_n z^{2n+1}$$ to the required degree of precision.

The arccosine function is derived from the arcsine using the substitution

$$acos(x) = \frac{\pi}{2} - asin(x)$$

The polynomial expansion of asin(z) is calculated in thirteen vector instructions, eleven of which are linked triads. The range of the results for arccosine is $\frac{-\pi}{2}$ to $+\frac{\pi}{2}$ inclusive, and for arccosine is 0 to π inclusive.

The following results were achieved, with times expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC aecsine	.5	.67	.87	1.27	2.6	4.73	9.64	29.84
64-bit vector arcsine	.52	.61	.75	1.04	2.02	3.55	6.69	16.54
32-bit vector arccosine	.54	.51	.58	.73	1.37	2.25	3.91	9.11

vector length	10	50	100	200	500	1000	2000	5000
CDC arccosine	.26	.68	.89	1.27	2.41	4.35	9.16	28.55
64-bit vector arccosine	.51	.61	.76	1.05	1.95	3.44	6.44	18.73
32-bit vector arccosine	.48	.54	.61	.76	1.25	2.07	3.66	8.59

Here our 32-bit functions are over three times as fast as the CDC routines, for vector lengths of 5000.

Conclusion

The trigonometrical and logarithmic functions, as provided by CDC up to and including version 2.0 of the compiler are, in general, not very efficient. At the Meteorological Office, we found it necessary to hand-code these functions in vector syntax to take full advantage of the facilities of the Cyber 205. For the 32-bit versions, which have a high enough precision for most of our purposes, speed increases of up to six times were obtained and even for our 64-bit versions,

increases of up to three times are possible. However, CDC have undertaken to provide fully vectorized versions of the trigonometrical andlogarithmic functions in both 64-bits and 32-bits by release 2.1 of the compiler.

The functions described were written in the "special call" syntax because of compiler limitations and the difficulties associated with this were partly offset by the special features which were then available. Users with the 2.0 compiler could find that the extra facilities provided by the "special calls" do not overcome the difficulties involved with this syntax and that coding explicitly in the FORTRAN vector syntax achieves sufficient vectorization for their own purposes.