Custom Math Functions for Molecular Dynamics

Robert Enenkel   Blake Fitch   Bob Germain
Fred Gustavson   Allan Martin   Mark Mendell   Jed Pitera
Mike Pitman   Alex Rayshubski   Frank Suits   Bill Swope
T J Chris Ward

28th April 2005

IBM T J Watson Research Centre, Yorktown Heights, New York

Abstract

While developing the protein folding application for IBM's BlueGene/L supercomputer, some frequently-executed computational kernels were encountered. These were significantly more complex than the linear algebra kernels that are normally provided as tuned libraries with modern machines. Using regular library functions for these would have resulted in an application which exploited only 5-10% of the potential floating-point throughput of the machine.

This is a tour of the functions encountered; they have been expressed in C++ (and could be expressed in other languages such as Fortran or C); with the help of a good optimising compiler, floating-point efficiency is much closer to 100%.

The implementations are offered in the hope that they may help in other implementations of Molecular Dynamics; in other fields of endeavour; and in the hope that others may adapt the ideas presented here to deliver additional mathematical functions at high throughput.

Contents

0.1 Custom Math Functions

0.1.1 Vectorisable Math Functions

The IBM XLC compiler can schedule instructions flexibly within a basic block, that is, a sequence of code with no conditional branches.

This document explains how to exploit this for functions commonly used in Molecular Dynamics; if you can enable the compiler to see enough independent work, it will schedule instructions to avoid stalls in the floating-point execution pipeline; and the hardware will run at a high fraction of peak throughput.

To exploit this, it is generally necessary to avoid special cases and error handling; all of these math functions will return a scalar result, will not set errno, and will not signal a NaN in any useful way. They can be wrapped to produce conventional results for out-of-domain cases; for example to produce NaN for $\log(-1)$; but for Molecular Dynamics we are generally confident that they will not be asked to process out-of-domain cases, and so the extra computation involved in getting conventional answers might best be skipped.

One way to let the compiler see independent work is to write it explicitly in the source code. Another way is to enclose the basic block in a counted loop and verify that the compiler can see that loop iterations are independent; then the compiler will apply loop transformation optimisations such as unrolling and modulo scheduling to construct the appropriate work itself.

0.1.2 Vectorisable log

log is vectorised by appreciating that a floating-point number is represented as an exponent and a mantissa; i.e. as $m \times 2^k$, for some m in [1.0, 2.0) and for integer k,

$$\ln(m \times 2^k) = \ln(m) + \ln(2^k)$$

The approximation is produced as three terms, which are added together to give the result.

k is extracted as the exponent part of the argument, giving the first term of the result as $k \times \ln(2)$

m is expressed as $m0 \times m1$, where $m0$ is $1 + \frac{a}{16}$ for integer a in (0, 15), and $m1$ is $\frac{m}{1+\frac{a}{16}}$.

a is determined by extracting the first 4 bits after the binary point from m.

$\frac{1}{1+\frac{a}{16}}$ is looked up in a 16-element table; and this gives a value for $m1$ roughly between 1 and $1 + \frac{1}{16}$.

The second term of the result is $\ln(m0)$, which comes from another 16-element table.

The third term of the result comes from a Taylor series for $\ln(1+x)$; this converges quite rapidly for $x < \frac{1}{16}$.

The full result is then

$$\ln(a) \simeq k \times \ln(2) + \text{Lookup}(a) + \text{TaylorSeries}(x)$$

An improvement comes from a slight modification, where $m1$ is arranged to be in the domain $[1-\frac{1}{32}, 1+\frac{1}{32})$, and so the Taylor series is used for $|x| < \frac{1}{32}$.

0.1.3 Vectorisable exp

exp is vectorised by appreciating that

$$\exp(a0+a1+a2+a3) = \exp(a0)\times\exp(a1)\times\exp(a2)\times\exp(a3)$$

$a0$ is extracted as the integer part of the argument. $a1$ is the next 4 bits; $a2$ is the subsequent 4 bits; and $a3$ is the remaining bits. $a3$ is a number between 0 and $\frac{1}{256}$.

$a0$ is shifted in to the exponent of the resulting floating-point number. $\exp(a1)$ and $\exp(a2)$ are looked up in 16-element tables. $\exp(a3)$ is estimated by a Taylor series, which converges quite rapidly for $0 < a3 < \frac{1}{256}$.

Again an improvement comes from a slight modification, setting a3 in the domain $[-\frac{1}{512}, +\frac{1}{512})$.

IBM PowerPC hardware supports a floating-point select instruction, which performs the equivalent of

```
double fsel(double a,double b, double c)
{
  if (a>=0.0) return b ; return c
}
```

as a single hardware instruction. This can be used to arrange that $\exp(x)$ returns 0 for a sufficiently-large negative argument,and Inf for a sufficiently-large positive argument, without causing a branch in the generated code.

0.1.4 Vectorisable erf/erfc - Piecewise Chebyshev

Traditionally in molecular dynamics codes, erfc(x) has been approximated by the formula in [1, Abramowitz and Stegun]

7.1.26

$$\operatorname{erf} x = 1-(a_1t+a_2t^2+a_3t^3+a_4t^4+a_5t^5)e^{-x^2}+\epsilon(x),$$

$$t=\frac{1}{1+px}$$

$$|\epsilon(x)|\le 1.5\times 10^{-7}$$

$p=.32759\ 11 \quad a_1=.25482\ 9592$

$a_2=-.28449\ 6736 \quad a_3=1.42141\ 3741$

$a_4=-1.45315\ 2027 \quad a_5=1.06140\ 5429$

Vectorisable $\exp(x)$ can be used to form vectorisable erfc in the obvious way; but there is an alternative which can be used to form a more accurate result. A more accurate result is desirable in Molecular Dynamics because it should give better energy conservation for a given time-step size; or alternatively will allow a larger time-step size before numerical instability sets in.

The reciprocal required above is a special case; for molecular dynamics codes, the dividend will be in the single-precision range, and there is no point returning a result much more accurate than the 1 part in 10^5 of the complete approximation. This leads to a faster expression of reciprocal than the hardware double-precision divide will give; more on this later.

For molecular dynamics, we are interested in erfc to support electrostatics, erfc(x) for a limited domain of x, typically $(-4, 4)$.

We partition the domain into equal-sized sub-domains, say $[-4, -3)$,$[-3, -2)$, .. , $[3, 4)$. Represent x as $x0 + x1$, where $x1$ is in $[-0.5, 0.5)$ and $x0$ is an integer which identifies the sub-domain. Each sub-domain is associated with a polynomial approximator; a set of 8 Chebyshev polynomials works well.

Select the appropriate polynomial by using $x0$ to index an array, and erfc(x) follows.

It is relatively easy to set the polynomials up to give erfc(x) accurate within 1 machine ulp (least significant bit) over the whole domain. It is desirable to use fsel to avoid travesties in case someone passes in a value of x outside the designed domain.

It is possible to exploit the symmetry between erfc(x) and erfc(-x) to halve the number of tables required.

The required table for Chebyshev coefficients is machine generated; [2, Numerical Recipes] shows the algorithm. First the Chebyshev coefficients for $\frac{d}{dx}\,\mathrm{erfc}(x)$are generated using the analytic expression $\frac{-2}{\sqrt{\pi}}\exp(-x^2)$; then the coefficients for $\mathrm{erfc}(x)$are generated by applying the appropriate transformation on these.

0.1.5 Vectorisable derivative erfc

Derivative erfc is $\frac{-2}{\sqrt{\pi}}\exp(-x^2)$, and may be vectorised using vectorisable $\exp(x)$.

However, for molecular dynamics,it is desirable to have derivative erfc and erfc related accurately as derivative and integral of each other; this results in better reported energy conservation, and better accuracy when switch or soft force cutoff is in use.

When the Abramowitz and Stegun approximation for $\mathrm{erfc}(x)$ is in use, we can differentiate the expression analytically. The derivative has an exponential term of the same form as the original, i.e. $\exp(-x^2)$, so a single evaluation of $\exp(X)$ will do duty for both functions when erfc and its derivative are both required in a computation.

When the multiple Chebyshev approach is in use, another set of Chebyshev polynomials can be used to deliver derivative erfc; if these are on the same sub-domains, there is a computational economy.

0.1.6 Vectorisable erfc and derivative - Piecewise Cubic Spline

In Molecular Dynamics, erfc and its derivative are used in the evaluation of electrostatic forces. Another approximation (particle mesh) means that it is not useful to get $\mathrm{erfc}(x)$ more precise than a relative error of about 10^{-5}; the imprecision due to the 'particle mesh' approximation dominates.

However, it is important for the values returned for $\mathrm{erfc}(x)$and its derivative to be continuous, and an analytic integral/derivative pair.

This can be satisfied by approximating $\frac{d}{dx}\,\mathrm{erfc}(x)$with a set of cubic splines, matching the$\frac{-2}{\sqrt{\pi}}\exp(-x^2)$ function and its derivative at the piecewise endpoints; and integrating these polynomials to give piecewise-quartic approximations for $\mathrm{erfc}(x)$.

A set of 64 piecewise cubic polynomials and their integrals, for domains [0,1/16), [1/16-2/16), .. , [63/16,64/16), gives the ability to approximate $\mathrm{erfc}(x)$and its derivative to the required precision in the domain [0-4).

0.1.7 Vectorisable sin/cos

It is convenient to use a multiple-Chebyshev-polynomial approach for this, too. Divide $\sin(x)$ into domains $[-45, 45)$,[45, 135),[135, 225), and [225, 315) degrees, and repeat cyclically.

In domains $[-45, 45)$ degrees and [135, 225) degrees, use a Chebyshev polynomial for $\frac{\sin(x)}{x}$, and multiply the result by x. This arranges that the result for small $|x|$ can be within an ulp, without requiring an excessive number of terms in the polynomial.

In domains [45, 135) and [225, 315), use a Chebyshev polynomial for $\cos(x)$.

The required Chebyshev polynomials are always even, which economises on the computation.

After the polynomial evaluation, fix up the result by a suitable multiply and add according to the sub-domain.

cos and sin are related since $\cos(x)$= $\sin(x + 90)$ with angles in degrees.

The tables are machine-generated offline, using extended-precision sinand cosfunctions and the algorithm in [2, Numerical Recipes].

0.1.8 Vectorisable inverse cos and sin

Sometimes an application will know the sin and cos of an angle, and will want to evaluate the angle. Traditional arcsin will involve an ambiguity as to the angle (80 degrees or 120 degrees, for example), is ill-conditioned in ranges near 90 and 270 degrees; and usually involves a conditional branch and a square root.

By expressing as

```
double acossin(double cos_angle, double sin_angle)
```

we can get over these limitations and produce an implementation without branches.

First, we take the absolute value of each of the parameters. Next, we use fsel to take whichever is smaller, and whichever is larger; giving a value between 0 and $\sqrt{0.5}$

representing the sin of an angle between 0 and 45 degrees, and a value between $\sqrt{0.5}$ and 1, representing the cos of the same angle.

Then we use the compound angle formula

$$\sin(a - b) = \sin(a)\cos(b) - \cos(a)\sin(b)$$

to form the sine of an angle in $[-22.5, 22.5)$ degrees, a value in the domain $[-0.38, 0.38)$ approximately.

Next, we use the Taylor expansion for $\arcsin(x)$ which converges quite rapidly over this domain; and we multiply by and add suitable constants (according as whether the original parameters were negated, and which was smaller) to evaluate the angle called for.

0.1.9 Vectorisable reciprocal square root

The natural way to express this is like

```
double a=1.0/sqrt(x);
```

With -qnostrict, the compiler knows about this. There is a hardware reciprocal square root estimate instruction which gives a result accurate to 5 bits (Power3) or 13 bits (BG/L) using lookup tables in the same amount of time that a multiply-add instruction would take; and the compiler generates a suitable number of iterations of Newton's method, or a suitable Taylor correction polynomial, to bring the result to double-precision accuracy.

Newton's iteration is expressed in terms of multiplies and adds; the 'divide by b' which seems to be required is replaced with 'multiply by estimate of $\frac{1}{b}$'.

0.1.10 Vectorisable square root

The compiler knows about

```
double a=sqrt(x)
```

and can use the hardware instruction for this on Power3, and generate a sequence something like $\frac{x}{\sqrt{x}}$ on BG/L. However, $\frac{x}{\sqrt{x}}$ on its own will give 'not-a-number' for $x = 0$; the compiler generates code to fix this up, but it is computationally expensive.

If the algorithm doesn't care about the result for $x = 0$, then it will run better on both Power3 and BG/L if coded as

```
double a=x/sqrt(x)
```

0.1.11 Vectorisable NearestImageInPeriodicVolume

Molecular dynamics is frequently run with 'periodic boundary conditions'; i.e. where we imagine that the simulation volume is surrounded by a never-ending sequence of matching simulation volumes; and the interaction force between a pair of atoms is calculated as if one of the atoms is influenced by the nearest of the 27 images of the other atom.

The 'nearest' image of an atom can be calculated without divisions or branches; map the simulation volume to a unit cube, by multiplying coordinates by the reciprocal of the simulation box. Then find the 'nearest integer' in each dimension, and subtract it off. Then multiply back up to the real simulation volume size.

0.1.12 Vectorisable nearest_integer

This relies on the IEEE floating-point representation. Double precision takes 64 bits. The top bit is a sign bit; the next 11 bits are a binary exponent; and the remaining 52 bits are a binary mantissa, with an implied leading '1'.

IEEE addition, with the hardware in its usual mode, is specified to round to the nearest integer. So if you take a double-precision floating-point number and add $(2^{52}+2^{51})$, the fractional part will be dropped. Then you can subtract the$(2^{52}+2^{51})$, and you will get the nearest integer to the number you started with.

There is a range around $2^{52}$where you will get the nearest even integer; so this is not applicable in all cases; but is OK for molecular dynamics.

The compiler is being asked to generate code for $(x + k) - k$; it is important to prevent the optimiser from re-associating this to $x + (k - k)$ and then optimising this to $x + 0$, i.e. x.

The sample code does this by expressing $(x+k \times k1) \times k1 - k$, where $k1$ is 1.0 but the compiler is unable to tell that $k1$ is a constant. IBM POWER family architectures support a 'multiply-add' instruction, so this does not cause any extra processing cycles.

0.1.13 Vectorisable 'Fragment In Range'

Molecular Dynamics is generally concerned with forces between atoms in an imagined simulation box with periodic boundary conditions. Computation of the force between a pair of atoms is skipped if the atoms are more than a threshold distance apart.

For computational convenience, the atoms are grouped into fragments; typically a water molecule, or a covalently-bonded set of atoms within a larger molecule. The question arises, 'given fragment a, what is the set of fragments $\{b_0, b_1, ...\}$such that an atom in a is in range of an atom in each b_i, accounting for the periodic boundary'. The simulation will be functionally correct if extra fragments b are in the set; the forces involved will evaluate to zero; but the simulation is more efficient with fewer extra fragments.

There is an algorithm for this which makes 100% use of the floating-point units, successively slicing for slab, cylinder, and sphere.

There is another algorithm which doesn't use the floating-point units; instead it uses the integer units with wrap at 2^{32}, successively slicing for slab, square prism, and cube; then uses the floating-point units to slice for sphere.

On Power3 and BG/L, the integer algorithm is faster; and either algorithm is sufficiently fast that the BlueMatter code does not need to maintain lists of fragments found to be in range in previous simulation time steps (known as Verlet lists) of fragments previously known to be in range, for system sizes of interest.

These algorithms show how to do 'vector compress'; i.e. producing a vector which is a subset of a starting vector, including only those elements matching a selection criterion, without requiring a conditional branch.

0.2 A practical example - reciprocal square roots

The function presented evaluates 'reciprocal square root' for each of 9 values, as would be needed to support the calculation of distances between atoms in a pair of 3-site water molecules.

Source code is given, then compiler intermediate code with cycle counts, then compiler assembly listing, for Power3 and BG/L machine architectures.

Values are copied into local variables, to make it clear to the compiler what is intended if the function is called with source and target overlapping in memory.

Power3 requires a vector of length at least 6 to keep the floating-point units fully busy on this algorithm; BG/L requires a vector of length 10. The compiler finds an optimal instruction sequence in each case; 100% floating-point utilisation for Power3, and 90% utilisation (4 'parallel' ops then a 'primary' op) for BG/L.

The 'reciprocal square root estimate' instruction of Power3 gives 5 bits of precision; that of BG/L gives 13 bits of precision. BG/L requires fewer follow-on instructions to converge the estimate to double precision. Power3 uses a Newton-Raphson algorithm for convergence; BG/L uses a Taylor expansion.

The theoretical peak rate for BG/L hardware is 10 double-precision square roots per 40 clock cycles; by enclosing similar code in a 'for' loop, it is possible to get the IBM VisualAge compiler to generate code which achieves within a few cycles of this rate.

0.2.1 Power3

```
VisualAge C++ for AIX Compiler (DEVELOPMENT/BETA) Version 6.0 ---
>>>>> OPTIONS SECTION <<<<<
IGNERRNO        THREADED        ARCH=PWR3       OPT=3           ALIAS=ANSI
ALIGN=NATURAL   NOROPTR         NODIRECTSTORAGE PREFETCH
FLOAT=NOHSFLT:NORNDSNGL:NOHSSNGL:MAF:NORRM:FOLD:NONANS:RSQRT:FLTINT:NOEMULATE
MAXMEM=-1       NOSTRICT        NOSTRICT_INDUCTION   TBTABLE=SMALL   LIST
SHOWINC=NOSYS:NOUSR             SOURCE          STATICINLINE    TMPLPARSE=NO
NOEH
>>>>> SOURCE SECTION <<<<<

            1 | #include <math.h>
            2 | void nineroot(double* f, const double* x)
            3 | {
            4 |    double x0 = x[0] ;
            5 |    double x1 = x[1] ;
            6 |    double x2 = x[2] ;
```

```
    double x3 = x[3] ;
    double x4 = x[4] ;
    double x5 = x[5] ;
    double x6 = x[6] ;
    double x7 = x[7] ;
    double x8 = x[8] ;
    double r0 = 1.0/sqrt(x0) ;
    double r1 = 1.0/sqrt(x1) ;
    double r2 = 1.0/sqrt(x2) ;
    double r3 = 1.0/sqrt(x3) ;
    double r4 = 1.0/sqrt(x4) ;
    double r5 = 1.0/sqrt(x5) ;
    double r6 = 1.0/sqrt(x6) ;
    double r7 = 1.0/sqrt(x7) ;
    double r8 = 1.0/sqrt(x8) ;
    f[0] = r0 ;
    f[1] = r1 ;
    f[2] = r2 ;
    f[3] = r3 ;
    f[4] = r4 ;
    f[5] = r5 ;
    f[6] = r6 ;
    f[7] = r7 ;
    f[8] = r8 ;
}
>>>>> OBJECT SECTION, OPTIMIZATION <<<<<
** Procedure List for Proc #   1: nineroot__FPdPCd End of Phase 3 **
    0:      HDR
    3:      BB_BEGIN    2 /    0
    0:      PROC     f,x,gr3,gr4
    0:      DIRCTIV  issue_cycle,0
    0:      LFLR     gr0=lr
    0:      DIRCTIV  issue_cycle,3
    0:      CALLNR   _savef18,gr1,fp18-fp31,lr"
    0:      DIRCTIV  issue_cycle,4
    0:      LLR      lr=gr0
    4:      LFL      fp1=(double)(gr4,0)
    5:      LFL      fp2=(double)(gr4,8)
    0:      DIRCTIV  issue_cycle,5
    6:      LFL      fp3=(double)(gr4,16)
    7:      LFL      fp4=(double)(gr4,24)
    0:      DIRCTIV  issue_cycle,6
    8:      LFL      fp5=(double)(gr4,32)
    9:      LFL      fp6=(double)(gr4,40)
    0:      DIRCTIV  issue_cycle,7
   10:      LFL      fp7=(double)(gr4,48)
```

```
 11:       LFL       fp8=(double)(gr4,56)
645:       FRSQRE    fp9=fp1
645:       FRSQRE    fp10=fp2
  0:       DIRCTIV   issue_cycle,8
 12:       LFL       fp21=(double)(gr4,64)
645:       L4A       gr4=.+CONSTANT_AREA(gr2,0)
645:       FRSQRE    fp11=fp3
645:       FRSQRE    fp12=fp4
  0:       DIRCTIV   issue_cycle,9
645:       FRSQRE    fp13=fp5
645:       FRSQRE    fp31=fp6
  0:       DIRCTIV   issue_cycle,10
645:       LFS       fp0=+CONSTANT_AREA(gr4,0)
645:       FRSQRE    fp30=fp7
645:       FRSQRE    fp29=fp8
  0:       DIRCTIV   issue_cycle,11
645:       MFL       fp20=fp9,fp9,fcr
645:       FRSQRE    fp28=fp21
  0:       DIRCTIV   issue_cycle,12
645:       MFL       fp27=fp10,fp10,fcr
645:       MFL       fp26=fp11,fp11,fcr
  0:       DIRCTIV   issue_cycle,13
645:       FMS       fp1=fp1,fp1,fp0,fcr
645:       FMS       fp2=fp2,fp2,fp0,fcr
  0:       DIRCTIV   issue_cycle,14
645:       FMS       fp3=fp3,fp3,fp0,fcr
645:       FMS       fp18=fp21,fp21,fp0,fcr
  0:       DIRCTIV   issue_cycle,15
645:       FMS       fp4=fp4,fp4,fp0,fcr
645:       MFL       fp25=fp12,fp12,fcr
  0:       DIRCTIV   issue_cycle,16
645:       FMS       fp5=fp5,fp5,fp0,fcr
645:       MFL       fp24=fp13,fp13,fcr
  0:       DIRCTIV   issue_cycle,17
645:       FMS       fp6=fp6,fp6,fp0,fcr
645:       FMS       fp7=fp7,fp7,fp0,fcr
  0:       DIRCTIV   issue_cycle,18
645:       MFL       fp23=fp31,fp31,fcr
645:       FMS       fp8=fp8,fp8,fp0,fcr
  0:       DIRCTIV   issue_cycle,19
645:       MFL       fp22=fp30,fp30,fcr
645:       MFL       fp21=fp29,fp29,fcr
  0:       DIRCTIV   issue_cycle,20
645:       FNMS      fp20=fp0,fp20,fp1,fcr
645:       MFL       fp19=fp28,fp28,fcr
  0:       DIRCTIV   issue_cycle,21
```

```
645:       FNMS     fp27=fp0,fp27,fp2,fcr
645:       FNMS     fp26=fp0,fp26,fp3,fcr
  0:       DIRCTIV  issue_cycle,22
645:       FNMS     fp25=fp0,fp25,fp4,fcr
645:       FNMS     fp24=fp0,fp24,fp5,fcr
  0:       DIRCTIV  issue_cycle,23
645:       FNMS     fp23=fp0,fp23,fp6,fcr
645:       FNMS     fp22=fp0,fp22,fp7,fcr
  0:       DIRCTIV  issue_cycle,24
645:       MFL      fp9=fp9,fp20,fcr
645:       FNMS     fp21=fp0,fp21,fp8,fcr
  0:       DIRCTIV  issue_cycle,25
645:       FNMS     fp19=fp0,fp19,fp18,fcr
645:       MFL      fp10=fp10,fp27,fcr
  0:       DIRCTIV  issue_cycle,26
645:       MFL      fp11=fp11,fp26,fcr
645:       MFL      fp12=fp12,fp25,fcr
  0:       DIRCTIV  issue_cycle,27
645:       MFL      fp13=fp13,fp24,fcr
645:       MFL      fp31=fp31,fp23,fcr
  0:       DIRCTIV  issue_cycle,28
645:       MFL      fp30=fp30,fp22,fcr
645:       MFL      fp29=fp29,fp21,fcr
  0:       DIRCTIV  issue_cycle,29
645:       MFL      fp27=fp9,fp9,fcr
645:       MFL      fp28=fp28,fp19,fcr
  0:       DIRCTIV  issue_cycle,30
645:       MFL      fp26=fp10,fp10,fcr
645:       MFL      fp25=fp11,fp11,fcr
  0:       DIRCTIV  issue_cycle,31
645:       MFL      fp24=fp12,fp12,fcr
645:       MFL      fp23=fp13,fp13,fcr
  0:       DIRCTIV  issue_cycle,32
645:       MFL      fp22=fp31,fp31,fcr
645:       MFL      fp21=fp30,fp30,fcr
  0:       DIRCTIV  issue_cycle,33
645:       FNMS     fp27=fp0,fp27,fp1,fcr
645:       MFL      fp19=fp29,fp29,fcr
  0:       DIRCTIV  issue_cycle,34
645:       MFL      fp20=fp28,fp28,fcr
645:       FNMS     fp26=fp0,fp26,fp2,fcr
  0:       DIRCTIV  issue_cycle,35
645:       FNMS     fp25=fp0,fp25,fp3,fcr
645:       FNMS     fp24=fp0,fp24,fp4,fcr
  0:       DIRCTIV  issue_cycle,36
645:       FNMS     fp23=fp0,fp23,fp5,fcr
```

```
645:       FNMS     fp22=fp0,fp22,fp6,fcr
  0:       DIRCTIV  issue_cycle,37
645:       FNMS     fp21=fp0,fp21,fp7,fcr
645:       FNMS     fp19=fp0,fp19,fp8,fcr
  0:       DIRCTIV  issue_cycle,38
645:       MFL      fp9=fp9,fp27,fcr
645:       FNMS     fp27=fp0,fp20,fp18,fcr
  0:       DIRCTIV  issue_cycle,39
645:       MFL      fp10=fp10,fp26,fcr
645:       MFL      fp11=fp11,fp25,fcr
  0:       DIRCTIV  issue_cycle,40
645:       MFL      fp12=fp12,fp24,fcr
645:       MFL      fp13=fp13,fp23,fcr
  0:       DIRCTIV  issue_cycle,41
645:       MFL      fp31=fp31,fp22,fcr
645:       MFL      fp30=fp30,fp21,fcr
  0:       DIRCTIV  issue_cycle,42
645:       MFL      fp26=fp9,fp9,fcr
645:       MFL      fp29=fp29,fp19,fcr
  0:       DIRCTIV  issue_cycle,43
645:       MFL      fp27=fp28,fp27,fcr
645:       MFL      fp20=fp10,fp10,fcr
  0:       DIRCTIV  issue_cycle,44
645:       MFL      fp21=fp11,fp11,fcr
645:       MFL      fp22=fp12,fp12,fcr
  0:       DIRCTIV  issue_cycle,45
645:       MFL      fp23=fp13,fp13,fcr
645:       MFL      fp24=fp31,fp31,fcr
  0:       DIRCTIV  issue_cycle,46
645:       MFL      fp25=fp30,fp30,fcr
645:       MFL      fp28=fp29,fp29,fcr
  0:       DIRCTIV  issue_cycle,47
645:       FNMS     fp19=fp0,fp26,fp1,fcr
645:       MFL      fp26=fp27,fp27,fcr
  0:       DIRCTIV  issue_cycle,48
645:       FNMS     fp20=fp0,fp20,fp2,fcr
645:       FNMS     fp21=fp0,fp21,fp3,fcr
  0:       DIRCTIV  issue_cycle,49
645:       FNMS     fp22=fp0,fp22,fp4,fcr
645:       FNMS     fp23=fp0,fp23,fp5,fcr
  0:       DIRCTIV  issue_cycle,50
645:       FNMS     fp24=fp0,fp24,fp6,fcr
645:       FNMS     fp25=fp0,fp25,fp7,fcr
  0:       DIRCTIV  issue_cycle,51
645:       FNMS     fp26=fp0,fp26,fp18,fcr
645:       FNMS     fp28=fp0,fp28,fp8,fcr
```

```
  0:       DIRCTIV  issue_cycle,52
645:       MFL      fp9=fp9,fp19,fcr
645:       MFL      fp10=fp10,fp20,fcr
  0:       DIRCTIV  issue_cycle,53
645:       MFL      fp11=fp11,fp21,fcr
645:       MFL      fp12=fp12,fp22,fcr
  0:       DIRCTIV  issue_cycle,54
645:       MFL      fp13=fp13,fp23,fcr
645:       MFL      fp31=fp31,fp24,fcr
  0:       DIRCTIV  issue_cycle,55
645:       MFL      fp30=fp30,fp25,fcr
645:       MFL      fp29=fp29,fp28,fcr
  0:       DIRCTIV  issue_cycle,56
645:       MFL      fp28=fp9,fp9,fcr
645:       MFL      fp27=fp27,fp26,fcr
  0:       DIRCTIV  issue_cycle,57
645:       MFL      fp26=fp10,fp10,fcr
645:       MFL      fp25=fp11,fp11,fcr
  0:       DIRCTIV  issue_cycle,58
645:       MFL      fp24=fp12,fp12,fcr
645:       MFL      fp23=fp13,fp13,fcr
  0:       DIRCTIV  issue_cycle,59
645:       MFL      fp22=fp31,fp31,fcr
645:       MFL      fp21=fp30,fp30,fcr
  0:       DIRCTIV  issue_cycle,60
645:       MFL      fp20=fp27,fp27,fcr
645:       MFL      fp19=fp29,fp29,fcr
  0:       DIRCTIV  issue_cycle,61
 31:       CONSUME  gr1,gr2,lr,gr13-gr31,fp14-fp31,cr[234],fsr,fcr,ctr
645:       FNMS     fp1=fp0,fp28,fp1,fcr
645:       FNMS     fp2=fp0,fp26,fp2,fcr
  0:       DIRCTIV  issue_cycle,62
645:       FNMS     fp3=fp0,fp25,fp3,fcr
645:       FNMS     fp4=fp0,fp24,fp4,fcr
  0:       DIRCTIV  issue_cycle,63
645:       FNMS     fp5=fp0,fp23,fp5,fcr
645:       FNMS     fp6=fp0,fp22,fp6,fcr
  0:       DIRCTIV  issue_cycle,64
645:       FNMS     fp7=fp0,fp21,fp7,fcr
645:       FNMS     fp8=fp0,fp19,fp8,fcr
  0:       DIRCTIV  issue_cycle,65
645:       MFL      fp1=fp9,fp1,fcr
645:       FNMS     fp0=fp0,fp20,fp18,fcr
  0:       DIRCTIV  issue_cycle,66
645:       MFL      fp2=fp10,fp2,fcr
645:       MFL      fp3=fp11,fp3,fcr
```

```
     0:      DIRCTIV  issue_cycle,67
   645:      MFL      fp4=fp12,fp4,fcr
   645:      MFL      fp5=fp13,fp5,fcr
     0:      DIRCTIV  issue_cycle,68
   645:      MFL      fp6=fp31,fp6,fcr
   645:      MFL      fp7=fp30,fp7,fcr
    22:      STFL     (double)(gr3,0)=fp1
     0:      DIRCTIV  issue_cycle,69
   645:      MFL      fp1=fp29,fp8,fcr
   645:      MFL      fp0=fp27,fp0,fcr
    23:      STFL     (double)(gr3,8)=fp2
    24:      STFL     (double)(gr3,16)=fp3
     0:      DIRCTIV  issue_cycle,70
    25:      STFL     (double)(gr3,24)=fp4
    26:      STFL     (double)(gr3,32)=fp5
     0:      DIRCTIV  issue_cycle,71
    27:      STFL     (double)(gr3,40)=fp6
    28:      STFL     (double)(gr3,48)=fp7
     0:      DIRCTIV  issue_cycle,72
    29:      STFL     (double)(gr3,56)=fp1
    30:      STFL     (double)(gr3,64)=fp0
     0:      FENCE
    31:      CALLF    _restf18
     3:      BB_END
     4:      BB_BEGIN    3 /    0
    31:      PEND
     4:      BB_END
** End of Procedure List for Proc #   1: nineroot__FPdPCd End of Phase 3 **
-qdebug=PLST3:CYCLES:PLST3:CYCLES:PLSTHUMM:HUMMDBG:RECIPF:MAXGRIDICULOUS:NEWSCHED1:NEWSCHED2
 GPR's set/used:   s-uu s--- ---- ----  ---- ---- ---- ----
 FPR's set/used:   ssss ssss ssss ss--  --ss ssss ssss ssss
 CCR's set/used:   ---- ----
     | 000000                          PDEF     nineroot(double *, const double *)
    0|                                 PROC     f,x,gr3,gr4
    0| 000000 mfspr    7C0802A6   1     LFLR     gr0=lr
    0| 000004 bl       4BFFFFFD   0     CALLNR   _savef18,gr1,fp18-fp31,lr"
    0| 000008 mtspr    7C0803A6   1     LLR      lr=gr0
    4| 00000C lfd      C8240000   1     LFL      fp1=(double)(gr4,0)
    5| 000010 lfd      C8440008   1     LFL      fp2=(double)(gr4,8)
    6| 000014 lfd      C8640010   1     LFL      fp3=(double)(gr4,16)
    7| 000018 lfd      C8840018   1     LFL      fp4=(double)(gr4,24)
    8| 00001C lfd      C8A40020   1     LFL      fp5=(double)(gr4,32)
    9| 000020 lfd      C8C40028   1     LFL      fp6=(double)(gr4,40)
   10| 000024 lfd      C8E40030   1     LFL      fp7=(double)(gr4,48)
   11| 000028 lfd      C9040038   1     LFL      fp8=(double)(gr4,56)
  645| 00002C frsqrte  FD200834   1     FRSQRE   fp9=fp1
```

```
645| 000030 frsqrte  FD401034   1      FRSQRE   fp10=fp2
 12| 000034 lfd      CAA40040   1      LFL      fp21=(double)(gr4,64)
645| 000038 lwz      80820004   1      L4A      gr4=.+CONSTANT_AREA(gr2,0)
645| 00003C frsqrte  FD601834   1      FRSQRE   fp11=fp3
645| 000040 frsqrte  FD802034   1      FRSQRE   fp12=fp4
645| 000044 frsqrte  FDA02834   1      FRSQRE   fp13=fp5
645| 000048 frsqrte  FFE03034   1      FRSQRE   fp31=fp6
645| 00004C lfs      C0040000   1      LFS      fp0=+CONSTANT_AREA(gr4,0)
645| 000050 frsqrte  FFC03834   1      FRSQRE   fp30=fp7
645| 000054 frsqrte  FFA04034   1      FRSQRE   fp29=fp8
645| 000058 fmul     FE890272   1      MFL      fp20=fp9,fp9,fcr
645| 00005C frsqrte  FF80A834   1      FRSQRE   fp28=fp21
645| 000060 fmul     FF6A02B2   1      MFL      fp27=fp10,fp10,fcr
645| 000064 fmul     FF4B02F2   1      MFL      fp26=fp11,fp11,fcr
645| 000068 fmsub    FC210838   1      FMS      fp1=fp1,fp1,fp0,fcr
645| 00006C fmsub    FC421038   1      FMS      fp2=fp2,fp2,fp0,fcr
645| 000070 fmsub    FC631838   1      FMS      fp3=fp3,fp3,fp0,fcr
645| 000074 fmsub    FE55A838   1      FMS      fp18=fp21,fp21,fp0,fcr
645| 000078 fmsub    FC842038   1      FMS      fp4=fp4,fp4,fp0,fcr
645| 00007C fmul     FF2C0332   1      MFL      fp25=fp12,fp12,fcr
645| 000080 fmsub    FCA52838   1      FMS      fp5=fp5,fp5,fp0,fcr
645| 000084 fmul     FF0D0372   1      MFL      fp24=fp13,fp13,fcr
645| 000088 fmsub    FCC63038   1      FMS      fp6=fp6,fp6,fp0,fcr
645| 00008C fmsub    FCE73838   1      FMS      fp7=fp7,fp7,fp0,fcr
645| 000090 fmul     FEFF07F2   1      MFL      fp23=fp31,fp31,fcr
645| 000094 fmsub    FD084038   1      FMS      fp8=fp8,fp8,fp0,fcr
645| 000098 fmul     FEDE07B2   1      MFL      fp22=fp30,fp30,fcr
645| 00009C fmul     FEBD0772   1      MFL      fp21=fp29,fp29,fcr
645| 0000A0 fnmsub   FE94007C   1      FNMS     fp20=fp0,fp20,fp1,fcr
645| 0000A4 fmul     FE7C0732   1      MFL      fp19=fp28,fp28,fcr
645| 0000A8 fnmsub   FF7B00BC   1      FNMS     fp27=fp0,fp27,fp2,fcr
645| 0000AC fnmsub   FF5A00FC   1      FNMS     fp26=fp0,fp26,fp3,fcr
645| 0000B0 fnmsub   FF39013C   1      FNMS     fp25=fp0,fp25,fp4,fcr
645| 0000B4 fnmsub   FF18017C   1      FNMS     fp24=fp0,fp24,fp5,fcr
645| 0000B8 fnmsub   FEF701BC   1      FNMS     fp23=fp0,fp23,fp6,fcr
645| 0000BC fnmsub   FED601FC   1      FNMS     fp22=fp0,fp22,fp7,fcr
645| 0000C0 fmul     FD290532   1      MFL      fp9=fp9,fp20,fcr
645| 0000C4 fnmsub   FEB5023C   1      FNMS     fp21=fp0,fp21,fp8,fcr
645| 0000C8 fnmsub   FE7304BC   1      FNMS     fp19=fp0,fp19,fp18,fcr
645| 0000CC fmul     FD4A06F2   1      MFL      fp10=fp10,fp27,fcr
645| 0000D0 fmul     FD6B06B2   1      MFL      fp11=fp11,fp26,fcr
645| 0000D4 fmul     FD8C0672   1      MFL      fp12=fp12,fp25,fcr
645| 0000D8 fmul     FDAD0632   1      MFL      fp13=fp13,fp24,fcr
645| 0000DC fmul     FFFF05F2   1      MFL      fp31=fp31,fp23,fcr
645| 0000E0 fmul     FFDE05B2   1      MFL      fp30=fp30,fp22,fcr
645| 0000E4 fmul     FFBD0572   1      MFL      fp29=fp29,fp21,fcr
```

```
645| 0000E8 fmul     FF690272   1      MFL      fp27=fp9,fp9,fcr
645| 0000EC fmul     FF9C04F2   1      MFL      fp28=fp28,fp19,fcr
645| 0000F0 fmul     FF4A02B2   1      MFL      fp26=fp10,fp10,fcr
645| 0000F4 fmul     FF2B02F2   1      MFL      fp25=fp11,fp11,fcr
645| 0000F8 fmul     FF0C0332   1      MFL      fp24=fp12,fp12,fcr
645| 0000FC fmul     FEED0372   1      MFL      fp23=fp13,fp13,fcr
645| 000100 fmul     FEDF07F2   1      MFL      fp22=fp31,fp31,fcr
645| 000104 fmul     FEBE07B2   1      MFL      fp21=fp30,fp30,fcr
645| 000108 fnmsub   FF7B007C   1      FNMS     fp27=fp0,fp27,fp1,fcr
645| 00010C fmul     FE7D0772   1      MFL      fp19=fp29,fp29,fcr
645| 000110 fmul     FE9C0732   1      MFL      fp20=fp28,fp28,fcr
645| 000114 fnmsub   FF5A00BC   1      FNMS     fp26=fp0,fp26,fp2,fcr
645| 000118 fnmsub   FF3900FC   1      FNMS     fp25=fp0,fp25,fp3,fcr
645| 00011C fnmsub   FF18013C   1      FNMS     fp24=fp0,fp24,fp4,fcr
645| 000120 fnmsub   FEF7017C   1      FNMS     fp23=fp0,fp23,fp5,fcr
645| 000124 fnmsub   FED601BC   1      FNMS     fp22=fp0,fp22,fp6,fcr
645| 000128 fnmsub   FEB501FC   1      FNMS     fp21=fp0,fp21,fp7,fcr
645| 00012C fnmsub   FE73023C   1      FNMS     fp19=fp0,fp19,fp8,fcr
645| 000130 fmul     FD2906F2   1      MFL      fp9=fp9,fp27,fcr
645| 000134 fnmsub   FF7404BC   1      FNMS     fp27=fp0,fp20,fp18,fcr
645| 000138 fmul     FD4A06B2   1      MFL      fp10=fp10,fp26,fcr
645| 00013C fmul     FD6B0672   1      MFL      fp11=fp11,fp25,fcr
645| 000140 fmul     FD8C0632   1      MFL      fp12=fp12,fp24,fcr
645| 000144 fmul     FDAD05F2   1      MFL      fp13=fp13,fp23,fcr
645| 000148 fmul     FFFF05B2   1      MFL      fp31=fp31,fp22,fcr
645| 00014C fmul     FFDE0572   1      MFL      fp30=fp30,fp21,fcr
645| 000150 fmul     FF490272   1      MFL      fp26=fp9,fp9,fcr
645| 000154 fmul     FFBD04F2   1      MFL      fp29=fp29,fp19,fcr
645| 000158 fmul     FF7C06F2   1      MFL      fp27=fp28,fp27,fcr
645| 00015C fmul     FE8A02B2   1      MFL      fp20=fp10,fp10,fcr
645| 000160 fmul     FEAB02F2   1      MFL      fp21=fp11,fp11,fcr
645| 000164 fmul     FECC0332   1      MFL      fp22=fp12,fp12,fcr
645| 000168 fmul     FEED0372   1      MFL      fp23=fp13,fp13,fcr
645| 00016C fmul     FF1F07F2   1      MFL      fp24=fp31,fp31,fcr
645| 000170 fmul     FF3E07B2   1      MFL      fp25=fp30,fp30,fcr
645| 000174 fmul     FF9D0772   1      MFL      fp28=fp29,fp29,fcr
645| 000178 fnmsub   FE7A007C   1      FNMS     fp19=fp0,fp26,fp1,fcr
645| 00017C fmul     FF5B06F2   1      MFL      fp26=fp27,fp27,fcr
645| 000180 fnmsub   FE9400BC   1      FNMS     fp20=fp0,fp20,fp2,fcr
645| 000184 fnmsub   FEB500FC   1      FNMS     fp21=fp0,fp21,fp3,fcr
645| 000188 fnmsub   FED6013C   1      FNMS     fp22=fp0,fp22,fp4,fcr
645| 00018C fnmsub   FEF7017C   1      FNMS     fp23=fp0,fp23,fp5,fcr
645| 000190 fnmsub   FF1801BC   1      FNMS     fp24=fp0,fp24,fp6,fcr
645| 000194 fnmsub   FF3901FC   1      FNMS     fp25=fp0,fp25,fp7,fcr
645| 000198 fnmsub   FF5A04BC   1      FNMS     fp26=fp0,fp26,fp18,fcr
645| 00019C fnmsub   FF9C023C   1      FNMS     fp28=fp0,fp28,fp8,fcr
```

```
645| 0001A0 fmul     FD2904F2   1     MFL       fp9=fp9,fp19,fcr
645| 0001A4 fmul     FD4A0532   1     MFL       fp10=fp10,fp20,fcr
645| 0001A8 fmul     FD6B0572   1     MFL       fp11=fp11,fp21,fcr
645| 0001AC fmul     FD8C05B2   1     MFL       fp12=fp12,fp22,fcr
645| 0001B0 fmul     FDAD05F2   1     MFL       fp13=fp13,fp23,fcr
645| 0001B4 fmul     FFFF0632   1     MFL       fp31=fp31,fp24,fcr
645| 0001B8 fmul     FFDE0672   1     MFL       fp30=fp30,fp25,fcr
645| 0001BC fmul     FFBD0732   1     MFL       fp29=fp29,fp28,fcr
645| 0001C0 fmul     FF890272   1     MFL       fp28=fp9,fp9,fcr
645| 0001C4 fmul     FF7B06B2   1     MFL       fp27=fp27,fp26,fcr
645| 0001C8 fmul     FF4A02B2   1     MFL       fp26=fp10,fp10,fcr
645| 0001CC fmul     FF2B02F2   1     MFL       fp25=fp11,fp11,fcr
645| 0001D0 fmul     FF0C0332   1     MFL       fp24=fp12,fp12,fcr
645| 0001D4 fmul     FEED0372   1     MFL       fp23=fp13,fp13,fcr
645| 0001D8 fmul     FEDF07F2   1     MFL       fp22=fp31,fp31,fcr
645| 0001DC fmul     FEBE07B2   1     MFL       fp21=fp30,fp30,fcr
645| 0001E0 fmul     FE9B06F2   1     MFL       fp20=fp27,fp27,fcr
645| 0001E4 fmul     FE7D0772   1     MFL       fp19=fp29,fp29,fcr
645| 0001E8 fnmsub   FC3C007C   1     FNMS      fp1=fp0,fp28,fp1,fcr
645| 0001EC fnmsub   FC5A00BC   1     FNMS      fp2=fp0,fp26,fp2,fcr
645| 0001F0 fnmsub   FC7900FC   1     FNMS      fp3=fp0,fp25,fp3,fcr
645| 0001F4 fnmsub   FC98013C   1     FNMS      fp4=fp0,fp24,fp4,fcr
645| 0001F8 fnmsub   FCB7017C   1     FNMS      fp5=fp0,fp23,fp5,fcr
645| 0001FC fnmsub   FCD601BC   1     FNMS      fp6=fp0,fp22,fp6,fcr
645| 000200 fnmsub   FCF501FC   1     FNMS      fp7=fp0,fp21,fp7,fcr
645| 000204 fnmsub   FD13023C   1     FNMS      fp8=fp0,fp19,fp8,fcr
645| 000208 fmul     FC290072   1     MFL       fp1=fp9,fp1,fcr
645| 00020C fnmsub   FC1404BC   1     FNMS      fp0=fp0,fp20,fp18,fcr
645| 000210 fmul     FC4A00B2   1     MFL       fp2=fp10,fp2,fcr
645| 000214 fmul     FC6B00F2   1     MFL       fp3=fp11,fp3,fcr
645| 000218 fmul     FC8C0132   1     MFL       fp4=fp12,fp4,fcr
645| 00021C fmul     FCAD0172   1     MFL       fp5=fp13,fp5,fcr
645| 000220 fmul     FCDF01B2   1     MFL       fp6=fp31,fp6,fcr
645| 000224 fmul     FCFE01F2   1     MFL       fp7=fp30,fp7,fcr
 22| 000228 stfd     D8230000   1     STFL      (double)(gr3,0)=fp1
645| 00022C fmul     FC3D0232   1     MFL       fp1=fp29,fp8,fcr
645| 000230 fmul     FC1B0032   1     MFL       fp0=fp27,fp0,fcr
 23| 000234 stfd     D8430008   1     STFL      (double)(gr3,8)=fp2
 24| 000238 stfd     D8630010   1     STFL      (double)(gr3,16)=fp3
 25| 00023C stfd     D8830018   1     STFL      (double)(gr3,24)=fp4
 26| 000240 stfd     D8A30020   1     STFL      (double)(gr3,32)=fp5
 27| 000244 stfd     D8C30028   1     STFL      (double)(gr3,40)=fp6
 28| 000248 stfd     D8E30030   1     STFL      (double)(gr3,48)=fp7
 29| 00024C stfd     D8230038   1     STFL      (double)(gr3,56)=fp1
 30| 000250 stfd     D8030040   1     STFL      (double)(gr3,64)=fp0
 31| 000254 b        4BFFFDAC   0     CALLF     _restf18
```

```
|               Tag Table
| 000258        00000000 00092200 0E000000 00000258
|               Instruction count          150
|               Constant Area
| 000000        3FC00000
```

0.2.2 BG/L

```
IBM(R) VisualAge C++ Version 6.0.0.3 for Linux on pSeries ---
>>>>> OPTIONS SECTION <<<<<
IGNERRNO        ARCH=440D       OPT=3           ALIAS=ANSI      ALIGN=LINUXPPC
FLOAT=NOHSFLT:NORNDSNGL:NOHSSNGL:MAF:NORRM:FOLD:NONANS:RSQRT:FLTINT:NOEMULATE
MAXMEM=-1       NOSTRICT        NOSTRICT_INDUCTION   TBTABLE=SMALL   LIST
SHOWINC=NOSYS:NOUSR             SOURCE          STATICINLINE    TMPLPARSE=NO
NOEH
>>>>> SOURCE SECTION <<<<<
            1 | #include <math.h>
            2 | void nineroot(double* f, const double* x)
            3 | {
            4 |    double x0 = x[0] ;
            5 |    double x1 = x[1] ;
            6 |    double x2 = x[2] ;
            7 |    double x3 = x[3] ;
            8 |    double x4 = x[4] ;
            9 |    double x5 = x[5] ;
           10 |    double x6 = x[6] ;
           11 |    double x7 = x[7] ;
           12 |    double x8 = x[8] ;
           13 |    double r0 = 1.0/sqrt(x0) ;
           14 |    double r1 = 1.0/sqrt(x1) ;
           15 |    double r2 = 1.0/sqrt(x2) ;
           16 |    double r3 = 1.0/sqrt(x3) ;
           17 |    double r4 = 1.0/sqrt(x4) ;
           18 |    double r5 = 1.0/sqrt(x5) ;
           19 |    double r6 = 1.0/sqrt(x6) ;
           20 |    double r7 = 1.0/sqrt(x7) ;
           21 |    double r8 = 1.0/sqrt(x8) ;
           22 |    f[0] = r0 ;
           23 |    f[1] = r1 ;
           24 |    f[2] = r2 ;
           25 |    f[3] = r3 ;
           26 |    f[4] = r4 ;
           27 |    f[5] = r5 ;
           28 |    f[6] = r6 ;
           29 |    f[7] = r7 ;
```

```
         30 |    f[8] = r8 ;
         31 | }
** Procedure List for Proc #   1: _Z8ninerootPdPKd End of Phase 3 **
    0:       HDR
    4:       BB_BEGIN    2 /    0
    3:       PROC      f,x,gr3,gr4
    0:       DIRCTIV  issue_cycle,0
    0:       LR       gr12=gr1
    0:       LI       gr0=-16
    0:       DIRCTIV  issue_cycle,1
    0:       ST4U     gr1,#stack(gr1,-96)=gr1
    0:       DIRCTIV  issue_cycle,2
    0:       SFPLU    gr12,#stack(gr12,gr0,0)=fp31,fp63
    0:       DIRCTIV  issue_cycle,3
    0:       SFPLU    gr12,#stack(gr12,gr0,0)=fp30,fp62
    0:       DIRCTIV  issue_cycle,4
    0:       SFPLU    gr12,#stack(gr12,gr0,0)=fp29,fp61
    0:       DIRCTIV  issue_cycle,5
    0:       SFPLU    gr12,#stack(gr12,gr0,0)=fp28,fp60
    0:       DIRCTIV  issue_cycle,6
    0:       SFPLU    gr12,#stack(gr12,gr0,0)=fp27,fp59
    0:       FENCE
    0:       DIRCTIV  end_prologue
    0:       FENCE
    0:       DIRCTIV  issue_cycle,0
   31:       DIRCTIV  start_epilogue
    4:       LFL      fp13=(double)(gr4,0)
    5:       LI       gr6=8
    0:       DIRCTIV  issue_cycle,1
    7:       LI       gr5=24
    5:       LFL      fp45=(double)(gr4,gr6,0,trap=8)
    0:       DIRCTIV  issue_cycle,2
    9:       LI       gr8=40
   11:       LI       gr6=56
    0:       DIRCTIV  issue_cycle,3
   13:       LA       gr7=.+CONSTANT_AREA%HI(gr2,0)
    6:       LFL      fp11=(double)(gr4,16)
    0:       DIRCTIV  issue_cycle,4
   13:       LA       gr7=+CONSTANT_AREA%LO(gr7,0)
    7:       LFL      fp43=(double)(gr4,gr5,0,trap=24)
    0:       DIRCTIV  issue_cycle,5
    8:       LFL      fp10=(double)(gr4,32)
   13:       FPRSQRE  fp9,fp41=fp13,fp45
    0:       DIRCTIV  issue_cycle,6
    9:       LFL      fp42=(double)(gr4,gr8,0,trap=40)
   31:       LR       gr12=gr1
```

```
 0:       DIRCTIV  issue_cycle,7
10:       LFL      fp8=(double)(gr4,48)
31:       LI       gr0=16
 0:       DIRCTIV  issue_cycle,8
11:       LFL      fp40=(double)(gr4,gr6,0,trap=56)
15:       FPRSQRE  fp7,fp39=fp11,fp43
 0:       DIRCTIV  issue_cycle,9
13:       LI       gr6=32
12:       LFL      fp31=(double)(gr4,64)
 0:       DIRCTIV  issue_cycle,10
13:       LFPS     fp27,fp59=+CONSTANT_AREA(gr7,gr5,0,trap=24)
13:       FPMUL    fp12,fp44=fp9,fp41,fp9,fp41,fcr
 0:       DIRCTIV  issue_cycle,11
17:       FPRSQRE  fp6,fp38=fp10,fp42
13:       LFS      fp30=+CONSTANT_AREA(gr7,4)
 0:       DIRCTIV  issue_cycle,12
19:       FPRSQRE  fp4,fp36=fp8,fp40
13:       LFPS     fp5,fp37=+CONSTANT_AREA(gr7,gr6,0,trap=32)
 0:       DIRCTIV  issue_cycle,13
21:       FRSQRE   fp29=fp31
13:       LFPS     fp3,fp35=+CONSTANT_AREA(gr7,gr8,0,trap=40)
 0:       DIRCTIV  issue_cycle,14
13:       LI       gr4=48
15:       FPMUL    fp1,fp33=fp7,fp39,fp7,fp39,fcr
 0:       DIRCTIV  issue_cycle,15
13:       LFPS     fp2,fp34=+CONSTANT_AREA(gr7,gr4,0,trap=48)
13:       FPMADD   fp13,fp45=fp27,fp59,fp13,fp45,fp12,fp44,fcr
 0:       DIRCTIV  issue_cycle,16
17:       FPMUL    fp0,fp32=fp6,fp38,fp6,fp38,fcr
23:       LI       gr6=8
 0:       DIRCTIV  issue_cycle,17
19:       FPMUL    fp12,fp44=fp4,fp36,fp4,fp36,fcr
 0:       DIRCTIV  issue_cycle,18
21:       MFL      fp28=fp29,fp29,fcr
 0:       DIRCTIV  issue_cycle,19
15:       FPMADD   fp1,fp33=fp27,fp59,fp11,fp43,fp1,fp33,fcr
 0:       DIRCTIV  issue_cycle,21
17:       FPMADD   fp10,fp42=fp27,fp59,fp10,fp42,fp0,fp32,fcr
 0:       DIRCTIV  issue_cycle,22
19:       FPMADD   fp8,fp40=fp27,fp59,fp8,fp40,fp12,fp44,fcr
 0:       DIRCTIV  issue_cycle,23
21:       FMA      fp31=fp27,fp31,fp28,fcr
 0:       DIRCTIV  issue_cycle,24
13:       FXPMADD  fp0,fp32=fp5,fp37,fp13,fp45,fp30,fp30,fcr
 0:       DIRCTIV  issue_cycle,25
15:       FXPMADD  fp12,fp44=fp5,fp37,fp1,fp33,fp30,fp30,fcr
```

```
31:       LFPLU    fp27,fp59,gr12=#stack(gr12,gr0,0)
 0:       DIRCTIV  issue_cycle,26
17:       FXPMADD  fp11,fp43=fp5,fp37,fp10,fp42,fp30,fp30,fcr
 0:       DIRCTIV  issue_cycle,27
19:       FXPMADD  fp28,fp60=fp5,fp37,fp8,fp40,fp30,fp30,fcr
 0:       DIRCTIV  issue_cycle,28
21:       FMA      fp5=fp5,fp31,fp30,fcr
 0:       DIRCTIV  issue_cycle,29
13:       FPMADD   fp0,fp32=fp3,fp35,fp13,fp45,fp0,fp32,fcr
 0:       DIRCTIV  issue_cycle,30
15:       FPMADD   fp12,fp44=fp3,fp35,fp1,fp33,fp12,fp44,fcr
 0:       DIRCTIV  issue_cycle,31
17:       FPMADD   fp11,fp43=fp3,fp35,fp10,fp42,fp11,fp43,fcr
 0:       DIRCTIV  issue_cycle,32
19:       FPMADD   fp30,fp62=fp3,fp35,fp8,fp40,fp28,fp60,fcr
 0:       DIRCTIV  issue_cycle,33
21:       FMA      fp5=fp3,fp31,fp5,fcr
 0:       DIRCTIV  issue_cycle,34
13:       FPMADD   fp0,fp32=fp2,fp34,fp13,fp45,fp0,fp32,fcr
31:       LFPLU    fp28,fp60,gr12=#stack(gr12,gr0,0)
 0:       DIRCTIV  issue_cycle,35
15:       FPMADD   fp3,fp35=fp2,fp34,fp1,fp33,fp12,fp44,fcr
 0:       DIRCTIV  issue_cycle,36
17:       FPMADD   fp11,fp43=fp2,fp34,fp10,fp42,fp11,fp43,fcr
 0:       DIRCTIV  issue_cycle,37
19:       FPMADD   fp12,fp44=fp2,fp34,fp8,fp40,fp30,fp62,fcr
 0:       DIRCTIV  issue_cycle,38
21:       FMA      fp5=fp2,fp31,fp5,fcr
 0:       DIRCTIV  issue_cycle,39
13:       FPMUL    fp0,fp32=fp13,fp45,fp0,fp32,fcr
 0:       DIRCTIV  issue_cycle,40
15:       FPMUL    fp1,fp33=fp1,fp33,fp3,fp35,fcr
 0:       DIRCTIV  issue_cycle,41
17:       FPMUL    fp2,fp34=fp10,fp42,fp11,fp43,fcr
 0:       DIRCTIV  issue_cycle,42
19:       FPMUL    fp3,fp35=fp8,fp40,fp12,fp44,fcr
 0:       DIRCTIV  issue_cycle,43
21:       MFL      fp5=fp31,fp5,fcr
 0:       DIRCTIV  issue_cycle,44
13:       FPMADD   fp0,fp32=fp9,fp41,fp9,fp41,fp0,fp32,fcr
 0:       DIRCTIV  issue_cycle,45
15:       FPMADD   fp1,fp33=fp7,fp39,fp7,fp39,fp1,fp33,fcr
 0:       DIRCTIV  issue_cycle,46
17:       FPMADD   fp2,fp34=fp6,fp38,fp6,fp38,fp2,fp34,fcr
 0:       DIRCTIV  issue_cycle,47
19:       FPMADD   fp3,fp35=fp4,fp36,fp4,fp36,fp3,fp35,fcr
```

```
     0:       DIRCTIV  issue_cycle,48
    21:       FMA      fp4=fp29,fp29,fp5,fcr
     0:       DIRCTIV  issue_cycle,49
    22:       STFL     (double)(gr3,0)=fp0
     0:       DIRCTIV  issue_cycle,50
    23:       STFL     (double)(gr3,gr6,0,trap=8)=fp32
    29:       LI       gr6=56
     0:       DIRCTIV  issue_cycle,51
    31:       LFPLU    fp29,fp61,gr12=#stack(gr12,gr0,0)
     0:       DIRCTIV  issue_cycle,52
    24:       STFL     (double)(gr3,16)=fp1
     0:       DIRCTIV  issue_cycle,53
    25:       STFL     (double)(gr3,gr5,0,trap=24)=fp33
     0:       DIRCTIV  issue_cycle,54
    31:       LFPLU    fp30,fp62,gr12=#stack(gr12,gr0,0)
     0:       DIRCTIV  issue_cycle,55
    26:       STFL     (double)(gr3,32)=fp2
     0:       DIRCTIV  issue_cycle,56
    27:       STFL     (double)(gr3,gr8,0,trap=40)=fp34
     0:       DIRCTIV  issue_cycle,57
    31:       LFPLU    fp31,fp63,gr12=#stack(gr12,gr0,0)
     0:       DIRCTIV  issue_cycle,58
    28:       STFL     (double)(gr3,48)=fp3
    31:       AI       gr1=gr1,96,gr12
     0:       DIRCTIV  issue_cycle,59
    31:       CONSUME  gr1,gr2,lr,gr14-gr31,fp14-fp31,fp46-fp63,cr[234],fsr,fcr,ctr
    29:       STFL     (double)(gr3,gr6,0,trap=56)=fp35
     0:       DIRCTIV  issue_cycle,60
    30:       STFL     (double)(gr3,64)=fp4
    31:       BA       lr
     4:       BB_END
     5:       BB_BEGIN    3 /    0
    31:       PEND
     5:       BB_END
** End of Procedure List for Proc #   1: _Z8ninerootPdPKd End of Phase 3 **
-qdebug=BGL:PLST3:CYCLES:SHUTUP:HUMMER:LINUX:NEWSCHED1:NEWSCHED2:REGPRES:ADRA:ANTIDEP:MAXGRI
 GPR's set/used:   ssuu ssss s--- s---  ---- ---- ---- ----
 FPR's set/used:   ssss ssss ssss ss--  ---- ---- ---s ssss
                   ssss ssss ssss ss--  ---- ---- ---s s-s-
 CCR's set/used:   ---- ----
  | 000000                    PDEF     nineroot(double *, const double *)
 3|                           PROC     f,x,gr3,gr4
 0| 000000 ori      602C0000 1 LR       gr12=gr1
 0| 000004 addi     3800FFF0 1 LI       gr0=-16
 0| 000008 stwu     9421FFA0 1 ST4U     gr1,#stack(gr1,-96)=gr1
 0| 00000C stfpdux  7FEC07DC 1 SFPLU    gr12,#stack(gr12,gr0,0)=fp31,fp63
```

```
 0| 000010 stfpdux  7FCC07DC 1 SFPLU    gr12,#stack(gr12,gr0,0)=fp30,fp62
 0| 000014 stfpdux  7FAC07DC 1 SFPLU    gr12,#stack(gr12,gr0,0)=fp29,fp61
 0| 000018 stfpdux  7F8C07DC 1 SFPLU    gr12,#stack(gr12,gr0,0)=fp28,fp60
 0| 00001C stfpdux  7F6C07DC 1 SFPLU    gr12,#stack(gr12,gr0,0)=fp27,fp59
 4| 000020 lfd      C9A40000 1 LFL      fp13=(double)(gr4,0)
 5| 000024 addi     38C00008 1 LI       gr6=8
 7| 000028 addi     38A00018 1 LI       gr5=24
 5| 00002C lfsdx    7DA4319C 1 LFL      fp45=(double)(gr4,gr6,0,trap=8)
 9| 000030 addi     39000028 1 LI       gr8=40
11| 000034 addi     38C00038 1 LI       gr6=56
13| 000038 addis    3CE00000 1 LA       gr7=.+CONSTANT_AREA%HI(gr2,0)
 6| 00003C lfd      C9640010 1 LFL      fp11=(double)(gr4,16)
13| 000040 addi     38E70000 1 LA       gr7=+CONSTANT_AREA%LO(gr7,0)
 7| 000044 lfsdx    7D64299C 1 LFL      fp43=(double)(gr4,gr5,0,trap=24)
 8| 000048 lfd      C9440020 1 LFL      fp10=(double)(gr4,32)
13| 00004C fprsqrte 0120681E 1 FPRSQRE  fp9,fp41=fp13,fp45
 9| 000050 lfsdx    7D44419C 1 LFL      fp42=(double)(gr4,gr8,0,trap=40)
31| 000054 ori      602C0000 1 LR       gr12=gr1
10| 000058 lfd      C9040030 1 LFL      fp8=(double)(gr4,48)
31| 00005C addi     38000010 1 LI       gr0=16
11| 000060 lfsdx    7D04319C 1 LFL      fp40=(double)(gr4,gr6,0,trap=56)
15| 000064 fprsqrte 00E0581E 1 FPRSQRE  fp7,fp39=fp11,fp43
13| 000068 addi     38C00020 1 LI       gr6=32
12| 00006C lfd      CBE40040 1 LFL      fp31=(double)(gr4,64)
13| 000070 lfpsx    7F672B1C 1 LFPS     fp27,fp59=+CONSTANT_AREA(gr7,gr5,0,trap=24)
13| 000074 fpmul    01890250 1 FPMUL    fp12,fp44=fp9,fp41,fp9,fp41,fcr
17| 000078 fprsqrte 00C0501E 1 FPRSQRE  fp6,fp38=fp10,fp42
13| 00007C lfs      C3C70004 1 LFS      fp30=+CONSTANT_AREA(gr7,4)
19| 000080 fprsqrte 0080401E 1 FPRSQRE  fp4,fp36=fp8,fp40
13| 000084 lfpsx    7CA7331C 1 LFPS     fp5,fp37=+CONSTANT_AREA(gr7,gr6,0,trap=32)
21| 000088 frsqrte  FFA0F834 1 FRSQRE   fp29=fp31
13| 00008C lfpsx    7C67431C 1 LFPS     fp3,fp35=+CONSTANT_AREA(gr7,gr8,0,trap=40)
13| 000090 addi     38800030 1 LI       gr4=48
15| 000094 fpmul    002701D0 1 FPMUL    fp1,fp33=fp7,fp39,fp7,fp39,fcr
13| 000098 lfpsx    7C47231C 1 LFPS     fp2,fp34=+CONSTANT_AREA(gr7,gr4,0,trap=48)
13| 00009C fpmadd   01ADDB20 1 FPMADD   fp13,fp45=fp27,fp59,fp13,fp45,fp12,fp44,fcr
17| 0000A0 fpmul    00060190 1 FPMUL    fp0,fp32=fp6,fp38,fp6,fp38,fcr
23| 0000A4 addi     38C00008 1 LI       gr6=8
19| 0000A8 fpmul    01840110 1 FPMUL    fp12,fp44=fp4,fp36,fp4,fp36,fcr
21| 0000AC fmul     FF9D0772 1 MFL      fp28=fp29,fp29,fcr
15| 0000B0 fpmadd   002BD860 1 FPMADD   fp1,fp33=fp27,fp59,fp11,fp43,fp1,fp33,fcr
17| 0000B4 fpmadd   014AD820 1 FPMADD   fp10,fp42=fp27,fp59,fp10,fp42,fp0,fp32,fcr
19| 0000B8 fpmadd   0108DB20 1 FPMADD   fp8,fp40=fp27,fp59,fp8,fp40,fp12,fp44,fcr
21| 0000BC fmadd    FFFFDF3A 1 FMA      fp31=fp27,fp31,fp28,fcr
13| 0000C0 fxcpmadd 001E2B64 1 FXPMADD  fp0,fp32=fp5,fp37,fp13,fp45,fp30,fp30,fcr
15| 0000C4 fxcpmadd 019E2864 1 FXPMADD  fp12,fp44=fp5,fp37,fp1,fp33,fp30,fp30,fcr
```

```
31| 0000C8 lfpdux   7F6C03DC 1 LFPLU    fp27,fp59,gr12=#stack(gr12,gr0,0)
17| 0000CC fxcpmadd 017E2AA4 1 FXPMADD  fp11,fp43=fp5,fp37,fp10,fp42,fp30,fp30,fcr
19| 0000D0 fxcpmadd 039E2A24 1 FXPMADD  fp28,fp60=fp5,fp37,fp8,fp40,fp30,fp30,fcr
21| 0000D4 fmadd    FCBF2FBA 1 FMA      fp5=fp5,fp31,fp30,fcr
13| 0000D8 fpmadd   000D1820 1 FPMADD   fp0,fp32=fp3,fp35,fp13,fp45,fp0,fp32,fcr
15| 0000DC fpmadd   01811B20 1 FPMADD   fp12,fp44=fp3,fp35,fp1,fp33,fp12,fp44,fcr
17| 0000E0 fpmadd   016A1AE0 1 FPMADD   fp11,fp43=fp3,fp35,fp10,fp42,fp11,fp43,fcr
19| 0000E4 fpmadd   03C81F20 1 FPMADD   fp30,fp62=fp3,fp35,fp8,fp40,fp28,fp60,fcr
21| 0000E8 fmadd    FCBF197A 1 FMA      fp5=fp3,fp31,fp5,fcr
13| 0000EC fpmadd   000D1020 1 FPMADD   fp0,fp32=fp2,fp34,fp13,fp45,fp0,fp32,fcr
31| 0000F0 lfpdux   7F8C03DC 1 LFPLU    fp28,fp60,gr12=#stack(gr12,gr0,0)
15| 0000F4 fpmadd   00611320 1 FPMADD   fp3,fp35=fp2,fp34,fp1,fp33,fp12,fp44,fcr
17| 0000F8 fpmadd   016A12E0 1 FPMADD   fp11,fp43=fp2,fp34,fp10,fp42,fp11,fp43,fcr
19| 0000FC fpmadd   018817A0 1 FPMADD   fp12,fp44=fp2,fp34,fp8,fp40,fp30,fp62,fcr
21| 000100 fmadd    FCBF117A 1 FMA      fp5=fp2,fp31,fp5,fcr
13| 000104 fpmul    000D0010 1 FPMUL    fp0,fp32=fp13,fp45,fp0,fp32,fcr
15| 000108 fpmul    002100D0 1 FPMUL    fp1,fp33=fp1,fp33,fp3,fp35,fcr
17| 00010C fpmul    004A02D0 1 FPMUL    fp2,fp34=fp10,fp42,fp11,fp43,fcr
19| 000110 fpmul    00680310 1 FPMUL    fp3,fp35=fp8,fp40,fp12,fp44,fcr
21| 000114 fmul     FCBF0172 1 MFL      fp5=fp31,fp5,fcr
13| 000118 fpmadd   00094820 1 FPMADD   fp0,fp32=fp9,fp41,fp9,fp41,fp0,fp32,fcr
15| 00011C fpmadd   00273860 1 FPMADD   fp1,fp33=fp7,fp39,fp7,fp39,fp1,fp33,fcr
17| 000120 fpmadd   004630A0 1 FPMADD   fp2,fp34=fp6,fp38,fp6,fp38,fp2,fp34,fcr
19| 000124 fpmadd   006420E0 1 FPMADD   fp3,fp35=fp4,fp36,fp4,fp36,fp3,fp35,fcr
21| 000128 fmadd    FC9DE97A 1 FMA      fp4=fp29,fp29,fp5,fcr
22| 00012C stfd     D8030000 1 STFL     (double)(gr3,0)=fp0
23| 000130 stfsdx   7C03359C 1 STFL     (double)(gr3,gr6,0,trap=8)=fp32
29| 000134 addi     38C00038 1 LI       gr6=56
31| 000138 lfpdux   7FAC03DC 1 LFPLU    fp29,fp61,gr12=#stack(gr12,gr0,0)
24| 00013C stfd     D8230010 1 STFL     (double)(gr3,16)=fp1
25| 000140 stfsdx   7C232D9C 1 STFL     (double)(gr3,gr5,0,trap=24)=fp33
31| 000144 lfpdux   7FCC03DC 1 LFPLU    fp30,fp62,gr12=#stack(gr12,gr0,0)
26| 000148 stfd     D8430020 1 STFL     (double)(gr3,32)=fp2
27| 00014C stfsdx   7C43459C 1 STFL     (double)(gr3,gr8,0,trap=40)=fp34
31| 000150 lfpdux   7FEC03DC 1 LFPLU    fp31,fp63,gr12=#stack(gr12,gr0,0)
28| 000154 stfd     D8630030 1 STFL     (double)(gr3,48)=fp3
31| 000158 addi     38210060 1 AI       gr1=gr1,96,gr12
29| 00015C stfsdx   7C63359C 1 STFL     (double)(gr3,gr6,0,trap=56)=fp35
30| 000160 stfd     D8830040 1 STFL     (double)(gr3,64)=fp4
31| 000164 bclr     4E800020 0 BA       lr
  |               Instruction count           90
  |               Constant Area
  | 000000        BF800000 3E8C0000 BEA00000 3EC00000 BF000000 49424D20
  | 000018        BF800000 BF800000 BEA00000 BEA00000 3EC00000 3EC00000
  | 000030        BF000000 BF000000
 1500-036: (I) The NOSTRICT option (default at OPT(3)) has the potential to
```

```
                alter the semantics of a program.
                Please refer to documentation on the STRICT/NOSTRICT option
                for more information.
```

0.3 Source code

0.3.1 Utilities

```
// 'fsel' is a built-in instruction on PPCGR and above,
// sometimes we want to force its use
#if defined(ARCH_HAS_FSEL)
#include <builtins.h>
#define fsel(a, x, y) __fsel((a),(x),(y))
#else
#define fsel(a, x, y) ( (a) >= 0.0 ? (x) : (y) )
#endif
/*
 * Storage mapping of an IEEE double-precision number, for access to
 * parts of it as integers or bits
 * This is big-endian specific, for little-endian you have to swap
 * m_hi and m_lo, then test it !
 * The intended use of this is in calculating exp(x)
 */
class DoubleMap
{
  public:
  class UIntPair
  {
     public:
    unsigned int m_hi ;
    unsigned int m_lo ;
  } ;
   union {
      double m_d ;
      UIntPair m_u ;
      } m_value ;
  DoubleMap(void) { } ;
  DoubleMap(double X) { m_value.m_d = X ; } ;
  DoubleMap(
    unsigned int Xsign ,   // 0 for positive, 1 for negative
    unsigned int Xexponent ,
    unsigned int Xsignificand_hi ,  // The 0x00100000 bit had
                    // better be set, to get the right answer
    unsigned int Xsignificand_lo
```

```
    ) {
       m_value.m_u.m_hi = ( ( Xsign << 31 )
                            & 0x80000000 )
                         | ( ( (Xexponent + 1023 )  << 20 )
                            & 0x7ff00000 )
                         | ( Xsignificand_hi
                            & 0x000fffff ) ;
       m_value.m_u.m_lo = Xsignificand_lo ;
   } ;
  double GetValue(void) const { return m_value.m_d ; } ;
  void SetValue(double X) { m_value.m_d = X ; } ;
  void SetValue(
    unsigned int Xsign ,   // 0 for positive, 1 for negative
    unsigned int Xexponent ,
    unsigned int Xsignificand_hi ,  // The 0x00100000 bit had better be
                                    // set, to get the right answer
    unsigned int Xsignificand_lo
    ) {
       m_value.m_u.m_hi = ( ( Xsign << 31 )
                            & 0x80000000 )
                         | ( ( (Xexponent + 1023 )  << 20 )
                            & 0x7ff00000 )
                         | ( Xsignificand_hi
                            & 0x000fffff ) ;
       m_value.m_u.m_lo = Xsignificand_lo ;
   } ;
  unsigned int HiWord(void) const { return m_value.m_u.m_hi ; } ;
  unsigned int LoWord(void) const { return m_value.m_u.m_lo ; } ;
  unsigned int SignBit(void) const { return HiWord() & 0x80000000 ; } ;
  unsigned int ExponentBits(void) const { return HiWord() & 0x7ff00000 ; } ;
  unsigned int SignificandHiBits(void) const { return HiWord() & 0x000fffff ; } ;
  unsigned int SignificandLoBits(void) const { return LoWord() ; } ;
  void SetSignificandHiBits(unsigned int new_hi_bits) {
                 m_value.m_u.m_hi = ( m_value.m_u.m_hi & 0xfff00000 )
                                    | ( new_hi_bits & 0x000fffff ) ; }
  int Exponent(void) const { return ( ExponentBits() >> 20 ) - 1023 ; } ;
  unsigned int SignificandHi(void) const {
             return SignificandHiBits() | 0x00100000 ; } ;
  unsigned int SignificandLo(void) const { return SignificandLoBits() ; } ;
  bool IsNegative(void) const { return  0 != SignBit() ; } ;
} ;
```

0.3.2 Nearest Integer

```
static double dk1 ; // The compiler does not know this is
```

```
  // constant, so should not 'optimise' away the rounding below
static inline double NearestInteger(const double x)
{
   const double two10 = 1024.0 ;
   const double two50 = two10 * two10 * two10 * two10 * two10 ;
   const double two52 = two50 * 4.0 ;
   const double two51 = two50 * 2.0 ;
   const double offset = two52 + two51 ;
   // Force add and subtract of appropriate constant
   // to drop  fractional part
   // .. hide it from the compiler so the optimiser won't
   // reassociate things ..
   const double losebits = (dk1*x) + offset ;
   const double result = (dk1*losebits) - offset ;
   return result ;
}
```

0.3.3 Reciprocal

This is useful for POWER3; the hardware 'floating-point divide' instruction blocks the floating-point pipeline and causes relatively low througput for code which is vectorisable. However, it only applies where the application is such that 'a' is in the single-precision range; the hardware 'fres' instruction does not give a useful result for double-precision numbers which cause overflow or underflow when converted to single precision.

For BG/L, the hardware 'parallel floating point reciprocal estimate' instruction gives a useful result for the whole double-precision range, and the compiler knows how to use it; this code sequence is therefore relatively less useful.

```
#include <builtins.h>
static inline double better_reciprocal(double a, double x0)
{
   double f0 = a*x0 - 1.0 ;
   double x1 = x0 - x0 * f0 ;
   return x1 ;
}
static inline double recip(double a)
{
   double x0 = __fres(a) ; // take it as read that a is in
                           // single-precision range
   double x1 = better_reciprocal(a,x0) ;
   double x2 = better_reciprocal(a,x1) ;
   double x3 = better_reciprocal(a,x2) ;
   return x3 ;
}
```

0.3.4 Complementary error function and derivative (Chebyshev)

```
class ChebyshevPairEvaluator
{
  public:
  enum {
     k_Terms = 16
     } ;
        class DoublePair
        {
                public:
                double pa ;
                double pb ;
        } ;
        class DoublePairArray
        {
                public:
                DoublePair c[k_Terms] ;
        } ;
  static void RawEvaluatePair(double& f, double& df, double x,
                              const DoublePairArray& cp)
  {
     double dppa = 0.0 ;
     double dpa = 0.0 ;
     double dppb = 0.0 ;
     double dpb = 0.0 ;
     for (int j=0; j<k_Terms-1; j+=1)
     {
        double da=(2.0*x)*dpa - dppa + cp.c[j].pa ;
        double db=(2.0*x)*dpb - dppb + cp.c[j].pb ;
        dppa = dpa ;
        dppb = dpb ;
        dpa = da ;
        dpb = db ;
     } /* endfor */
     // Term 0 is a special case; POWER 'multiply-add' makes
     // this same efficiency as rewriting the table
     double resulta = x*dpa - dppa + 0.5*(cp.c[k_Terms-1]).pa ;
     double resultb = x*dpb - dppb + 0.5*(cp.c[k_Terms-1]).pb ;
     f = resulta ;
     df = resultb ;
  }
} ;
class ErfEvaluator: public ChebyshevPairEvaluator
{
   public:
```

```
   enum {
      k_Slices = 8 ,
      } ;
        class CTable {
                public:
                DoublePairArray SliceTable[k_Slices] ;
        } ;
        static const CTable ChebyshevTable ;
        static double dk1 ; // The compiler does not know this is
         // constant, so should not 'optimise' away the rounding below
        static inline double NearestInteger(const double x)
        {
           const double two10 = 1024.0 ;
           const double two50 = two10 * two10 * two10 * two10 * two10 ;
           const double two52 = two50 * 4.0 ;
           const double two51 = two50 * 2.0 ;
           const double offset = two52 + two51 ;
           // Force add and subtract of appropriate constant to drop
           // fractional part
           // .. hide it from the compiler so the optimiser won't
           // reassociate things ..
           const double losebits = (dk1*x) + offset ;
           const double result = (dk1*losebits) - offset ;
           return result ;
        }
        static void Evaluate(const double x, double& f, double& df)
        {
           double xam = fabs(x) - 0.5 ;
           double xi = NearestInteger(xam) ;
           double xf = xam-xi ;                 // -0.5 < x < 0.5
           int ix = (int) xi ;
           int ixmask = ix & 7 ;
           double r0 ;
           double dr0 ;
           RawEvaluatePair(r0, dr0, 2.0*xf,
                           ChebyshevTable.SliceTable[ixmask]) ;
           double  r1 = fsel(xi-7.5,-1.0,  r0) ;
           double dr1 = fsel(xi-7.5, 0.0, dr0) ;
           double   m = fsel(x,1.0, -1.0) ;
           double  r2 = 1.0+m*r1 ;
           f = r2    ;
           df = dr1 ;
        }
} ;
#ifndef MSD_COMPILE_DATA_ONLY
const ErfEvaluator::CTable ErfEvaluator::ChebyshevTable = {
```

```
        {
                {
 {  {          -1.9514453114346613e-14 ,
               -4.3962027206143962e-14 }
,   {           1.3625665209647924e-13 ,
               -1.1610210709915711e-12 }
,   {           2.229902591391045e-12 ,
                7.5864104901966937e-12 }
,   {          -2.1427408195614467e-11 ,
                1.1479391368134277e-10 }
,   {          -2.0634390485502084e-10 ,
               -1.0209291828992976e-09 }
,   {           2.7257129691783295e-09 ,
               -8.9643378999395747e-09 }
,   {           1.4115809274810406e-08 ,
                1.0800758958423388e-07 }
,   {          -2.7818478883356451e-07 ,
                4.9920479599323502e-07 }
,   {          -5.2333748523420065e-07 ,
               -8.7939056530898302e-06 }
,   {           2.211114704099522e-05 ,
               -1.4154244790564384e-05 }
,   {          -1.8363892921493974e-05 ,
                0.00052187362333079546 }
,   {           -0.0012919410465849694 ,
               -0.00038143210322044387 }
,   {             0.004492934887683828 ,
                -0.020149183122028715 }
,   {             0.049552626796204341 ,
                0.053533786548985489 }
,   {            -0.42582445804381047 ,
                  0.37627183124760605 }
,   {            -0.93926583578230194 ,
                  -1.6497640456262563 }
} }
,{ {  {        -5.0317730474937121e-16 ,
               -4.2117485562976088e-14 }
,     {         9.4040069643033129e-14 ,
               -2.7959038508193287e-14 }
,     {        -5.4676286926192122e-13 ,
                5.2241264144468789e-12 }
,     {        -7.8746039930688689e-12 ,
               -2.8459628240128098e-11 }
,     {         1.1428281965722648e-10 ,
               -3.7275686525285883e-10 }
,     {         7.5247753102189958e-11 ,
```

```
                      4.999984436677837e-09 }
,     {            -1.1752407126361825e-08 ,
                     2.6371532588347393e-09 }
,     {             7.3301093509556885e-08 ,
                    -4.1808667211234786e-07 }
,     {             4.6810084713974775e-07 ,
                     2.3482721455646551e-06 }
,     {            -8.8537423943830358e-06 ,
                    1.2688737047800589e-05 }
,     {             3.2638635138919629e-05 ,
                    -0.00021014154531962822 }
,     {             0.00029384471334866456 ,
                    0.00066546143982619316 }
,     {             -0.0041797192715572768 ,
                     0.0044913738682590045 }
,     {               0.023555662412541055 ,
                     -0.049491169818861133 }
,     {              -0.072164111860376176 ,
                       0.19293667316858745 }
,     {                -1.8857045114727804 ,
                      -0.33814761726036585 }
} }
,{ {  {             1.1536030588919383e-15 ,
                    -4.5290945318974183e-15 }
,     {            -2.5831909518634811e-15 ,
                     6.9035600532037278e-14 }
,     {            -1.2219969549456367e-13 ,
                  -1.4918778783625236e-13 }
,     {               1.650088654139529e-12 ,
                  -6.2853485651852727e-12 }
,     {            -4.1430599274545238e-12 ,
                    7.9055067610861132e-11 }
,     {             -1.303276091975957e-10 ,
                  -1.8857998537318434e-10 }
,     {             1.8722149256193656e-09 ,
                  -5.1340493002929665e-09 }
,     {            -9.3478974795511597e-09 ,
                     6.721115733692397e-08 }
,     {            -4.9199521904598378e-08 ,
                  -3.0426676864593005e-07 }
,     {             1.2964140790298608e-06 ,
                    -1.3103754559918305e-06 }
,     {            -1.2343188523307585e-05 ,
                    3.0809671128070729e-05 }
,     {             7.5321057602619065e-05 ,
                  -0.00024817414592214355 }
```

```
,      {          -0.00032191106470368769 ,
                   0.0012359465927699756 }
,      {           0.0009701840770401455 ,
                  -0.0041111069223663957 }
,      {          -0.0019935206572262573 ,
                   0.0089974192090911398 }
,      {             -1.9973937586662434 ,
                  -0.012085189551271426 }
} }
,{ {  {            2.8811019587551389e-18 ,
                    2.684646603916068e-16 }
,      {          -6.2333922975783146e-16 ,
                   1.4252468434090293e-16 }
,      {           9.0877705975533128e-15 ,
                  -3.4638532206046957e-14 }
,      {          -6.2053772915893235e-14 ,
                   4.7270659575711313e-13 }
,      {          -9.4369582503557563e-14 ,
                 -3.0132196321689225e-12 }
,      {           8.1568391992615411e-12 ,
                  -3.6795550343994194e-12 }
,      {          -1.1859203571949966e-10 ,
                   3.2326034833829272e-10 }
,      {           1.1302927297982895e-09 ,
                  -4.2729928409363866e-09 }
,      {          -8.1653674229686321e-09 ,
                    3.649262770188356e-08 }
,      {           4.6711146297670056e-08 ,
                 -2.3290328068405808e-07 }
,      {          -2.1434786976693524e-07 ,
                   1.157560138845965e-06 }
,      {           7.8712065435992157e-07 ,
                 -4.5198606760227629e-06 }
,      {          -2.2827106290462468e-06 ,
                  1.3751490608604711e-05 }
,      {            5.108210075437179e-06 ,
                 -3.1912388224577725e-05 }
,      {           -8.532197326791333e-06 ,
                  5.4617171212102142e-05 }
,      {             -1.9999897804462703 ,
               -6.6041177531743063e-05 }
} }
,{ {  {          -7.4993123127081172e-19 ,
                 -9.2154967085056884e-19 }
,      {           1.6453122028975649e-17 ,
                  -4.4526109981603194e-17 }
```

```
,     {          -2.2216971245348065e-16 ,
                  9.2045328395178569e-16 }
,     {           2.3187986651361246e-15 ,
                -1.1597351157562596e-14 }
,     {          -1.9966897966189304e-14 ,
                 1.1222789210485 76e-13 }
,     {           1.4567188498750867e-13 ,
                 -8.9014086166989192e-13 }
,     {          -9.1094290960037775e-13 ,
                  5.9390981887108318e-12 }
,     {           4.9015331385524046e-12 ,
                 -3.3684085607283493e-11 }
,     {          -2.2668130864064367e-11 ,
                  1.6278815862238778e-10 }
,     {           8.9656333841687577e-11 ,
                 -6.6839174980108575e-10 }
,     {          -3.0070122824813192e-10 ,
                  2.3145401708228895e-09 }
,     {           8.4494270776318715e-10 ,
                 -6.6824163147637241e-09 }
,     {          -1.9576794331996479e-09 ,
                  1.5833623495033884e-08 }
,     {           3.6665887957624338e-09 ,
                 -3.0174569513159495e-08 }
,     {          -5.4258802952106441e-09 ,
                  4.5166333861133352e-08 }
,     {              -1.9999999937936801 ,
                 -5.1878090694002071e-08 }
} }
,{ {  {          -7.3128123766095007e-21 ,
                  5.2783768964163724e-20 }
,     {           5.8331678204505884e-20 ,
                 -4.4428465693890153e-19 }
,     {          -4.1665340361792927e-19 ,
                  3.3193577484164931e-18 }
,     {           2.6702440438238306e-18 ,
                 -2.2110261645071223e-17 }
,     {          -1.5352877622533002e-17 ,
                  1.3149107185196035e-16 }
,     {           7.9054987528274155e-17 ,
                 -6.9763687703652322e-16 }
,     {          -3.6338645128878323e-16 ,
                  3.2936905729829264e-15 }
,     {           1.4841921457728194e-15 ,
                 -1.3779549123432719e-14 }
,     {          -5.3535076599248966e-15 ,
```

```
                    5.0787839237713144e-14 }
,      {            1.6922815522375102e-14 ,
                   -1.6367776360132983e-13 }
,      {           -4.6440786423613345e-14 ,
                    4.5693541177471563e-13 }
,      {            1.0939887392901224e-13 ,
                   -1.0924934920735968e-12 }
,      {           -2.1831634568207554e-13 ,
                    2.2073173946389116e-12 }
,      {            3.6368947573372969e-13 ,
                   -3.7122896402585035e-12 }
,      {           -4.9822933420386177e-13 ,
                    5.1168332005087491e-12 }
,      {               -1.9999999999994464 ,
                   -5.7052069770739506e-12 }
} }
,{ {   {           -1.1581263794211803e-24 ,
                    1.1756024314709261e-23 }
,      {            6.8651109138591869e-24 ,
                   -7.1268610401803094e-23 }
,      {            -3.738518343124306e-23 ,
                    3.9620223549082376e-22 }
,      {            1.8656852708023271e-22 ,
                   -2.0152981488264423e-21 }
,      {           -8.5056522662053255e-22 ,
                    9.3514915353419952e-21 }
,      {            3.5290018709287991e-21 ,
                   -3.9440168120129876e-20 }
,      {           -1.3264545171699188e-20 ,
                   1.5051156637249394e-19 }
,      {            4.4925669595385181e-20 ,
                   -5.1696379430130061e-19 }
,      {           -1.3624823721052732e-19 ,
                    1.5881329934248198e-18 }
,      {            3.6731865126752203e-19 ,
                   -4.3319144361960655e-18 }
,      {           -8.7299974058134151e-19 ,
                    1.0403780623845348e-17 }
,      {            1.8120620323788916e-18 ,
                   -2.1791909247822896e-17 }
,      {            -3.251392893971571e-18 ,
                   3.9396773141907617e-17 }
,      {            4.9898793744105705e-18 ,
                   -6.0808623975481749e-17 }
,      {           -6.4850528855079053e-18 ,
                    7.9315808137192181e-17 }
```

```
,     {              -2.0000000000000009 ,
                 -8.6748835517513367e-17 }
} }
,{ {  {          -1.286993020143186e-29 ,
                  1.6244908657170241e-28 }
,     {           6.2765862086305583e-29 ,
                 -8.0275046304822356e-28 }
,     {          -2.839692427061988e-28 ,
                  3.6773373634048155e-27 }
,     {           1.1882092616181199e-27 ,
                 -1.5569151083770561e-26 }
,     {          -4.582191637967378e-27 ,
                  6.0711381921074574e-26 }
,     {           1.6221432712231815e-26 ,
                 -2.1718558315433516e-25 }
,     {          -5.247933806259193e-26 ,
                  7.0956869041034713e-25 }
,     {           1.5437131529880892e-25 ,
                 -2.1064417534076447e-24 }
,     {          -4.1053336780396758e-25 ,
                  5.6494507799722326e-24 }
,     {            9.807876328051367e-25 ,
                 -1.3601376051918737e-23 }
,     {          -2.0903055414542895e-24 ,
                  2.9188353967295514e-23 }
,     {           3.9443914370892561e-24 ,
                 -5.5407486881004531e-23 }
,     {          -6.5385165651342071e-24 ,
                  9.2298616960723605e-23 }
,     {           9.4486116746898621e-24 ,
                 -1.3386968566261501e-22 }
,     {          -1.1822408135528828e-23 ,
                  1.6788751035824249e-22 }
,     {              -2.0000000000000009 ,
                 -1.8115931820473034e-22 }
} }
                }
        }
 ;
```

0.3.5 Complementary error function and derivative (Spline)

```
class ErfEvaluator
{
   public:
   class doublepair
```

```
   {
           public:
           double ca ;
           double cb ;
   } ;
   class polypairlist
   {
           public:
           enum {
              k_Terms = 4
           } ;
           doublepair dp[k_Terms] ;
   } ;
   enum {
           k_Slices = 64
   } ;
   static const polypairlist p[k_Slices] ;
   static const float WholeNumbers[k_Slices] ;
   static const double IntegrationConstants[k_Slices] ;
   static void Evaluate (double xRaw, double& f, double& df)
   {
      assert(xRaw >= 0.0) ;                // This version only
                                   // defined for positive argument
      xRaw = fsel(xRaw-3.999,3.999,xRaw) ; // Pinned if out of range
      double xScale = xRaw * 16.0 ;
      int a = xScale ;
      double xWhole = WholeNumbers[a] ;
      double xFrac = xScale-xWhole ;
      double x = (xFrac*2.0) - 1.0 ;
      double rf =
((((p[a].dp[0].ca)*x+p[a].dp[1].ca)*x+p[a].dp[2].ca)*x+p[a].dp[3].ca) ;
      double rdf =
((((p[a].dp[0].cb)*x+p[a].dp[1].cb)*x+p[a].dp[2].cb)*x+p[a].dp[3].cb) ;
      f = rf * x + IntegrationConstants[a];
      df = rdf  ;
   }
} ;
#ifndef MSD_COMPILE_DATA_ONLY
const float ErfEvaluator::WholeNumbers[ErfEvaluator::k_Slices] =
{
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0
        ,10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0
        ,20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0
        ,30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0
        ,40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0
        ,50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0
```

```
        ,60.0, 61.0, 62.0, 63.0
} ;
const ErfEvaluator::polypairlist ErfEvaluator::p[ErfEvaluator::k_Slices] = {
 { {
  { -1.677043705606067e-08,-2.146615943175766e-06 }
  ,{ 1.143371616236766e-05,0.001097636751587296 }
  ,{ 3.440177110943935e-05,0.002201713351004118 }
  ,{ -0.03522741365975122,  -1.127277237112039 }
} }
,{ {
  { -4.965987042014532e-08,-6.356463413778601e-06 }
  ,{ 1.116730250618423e-05,0.001072061040593686 }
  ,{ 0.0001024021637152795, 0.00655373847777789 }
  ,{ -0.03495327227036313,   -1.11850471265162 }
} }
,{ {
  { -8.063031663779123e-08,-1.032068052963728e-05 }
  ,{ 1.06447944258563e-05,0.001021900264882204 }
  ,{ 0.000168024277188707, 0.01075355374007725 }
  ,{ -0.03441137302506241,  -1.101163936801997 }
} }
,{ {
  { -1.085177767547462e-07,-1.389027542460751e-05 }
  ,{ 9.886272248044382e-06,0.0009490821358122607 }
  ,{ 0.0002297848019711107, 0.01470622732615109 }
  ,{  -0.0336142358925829,  -1.075655548562653 }
} }
,{ {
  { -1.323228464452476e-07,-1.693732434499169e-05 }
  ,{ 8.920499832118013e-06,0.0008563679838833292 }
  ,{ 0.0002863479486297409, 0.01832626871230342 }
  ,{ -0.03258003643662622,  -1.042561165972039 }
} }
,{ {
  { -1.512604951606179e-07,-1.936134338055909e-05 }
  ,{ 7.783401736190406e-06,0.0007472065666742789 }
  ,{ 0.0003365732792269584, 0.02154068987052534 }
  ,{ -0.03133191701999147,  -1.002621344639727 }
} }
,{ {
  { -1.647954146847001e-07,-2.109381307964162e-05 }
  ,{ 6.51619848252759e-06,0.0006255550543226487 }
  ,{ 0.0003795532894002569, 0.02429141052161644 }
  ,{ -0.02989712649661543, -0.9567080478916936 }
} }
,{ {
```

```
  { -1.726609582696268e-07,-2.210060265851223e-05 }
  ,{ 5.163323960181864e-06,0.0004956791001774589 }
  ,{ 0.0004146390499898948, 0.02653689919935327 }
  ,{ -0.02830603301975991,  -0.905793056632317 }
} }
,{ {
  { -1.748610042477048e-07,-2.238220854370622e-05 }
  ,{ 3.77025961848157e-06,0.0003619449233742307 }
  ,{ 0.0004414530010017535, 0.0282529920641123 }
  ,{ -0.02659105981260158, -0.8509139140032504 }
} }
,{ {
  { -1.716553961417857e-07,-2.197189070614856e-05 }
  ,{ 2.381420131299626e-06,0.0002286163326047641 }
  ,{ 0.0004598888066024617, 0.02943288362255755 }
  ,{ -0.02478559574453316, -0.7931390638250611 }
} }
,{ {
  { -1.63530815350447e-07,-2.093194436485722e-05 }
  ,{ 1.03821508918815e-06,9.966864856206237e-05 }
  ,{ 0.000470098964779177, 0.03008633374586733 }
  ,{ -0.02292293133148951, -0.7335338026076644 }
} }
,{ {
  { -1.511599296296717e-07,-1.934847099259798e-05 }
  ,{ -2.22607659821702e-07,-2.137033534288339e-05 }
  ,{ 0.0004724715617529516,  0.0302381799521889 }
  ,{ -0.02103526657315815, -0.6731285303410608 }
} }
,{ {
  { -1.353523545145142e-07,-1.732510137785782e-05 }
  ,{ -1.370257092802229e-06,-0.000131544680909014 }
  ,{ 0.0004675981220443889, 0.02992627981084089 }
  ,{ -0.01915283030640111, -0.6128905698048354 }
} }
,{ {
  { -1.170013221461676e-07,-1.497616923470946e-05 }
  ,{ -2.380782501521628e-06,-0.0002285551201460763 }
  ,{ 0.0004562348970672072, 0.02919903341230126 }
  ,{ -0.01730314210665839, -0.5537005474130686 }
} }
,{ {
  { -9.702996492746713e-08,-1.241983551071579e-05 }
  ,{ -3.237526760163412e-06,-0.0003108025689756875 }
  ,{ 0.0004392601411388399, 0.02811264903288575 }
  ,{ -0.01551043792261867, -0.4963340135237975 }
```

```
} }
,{ {
  { -7.634081565236948e-08,-9.771624403503293e-06 }
  ,{ -3.931160936497558e-06,-0.0003773914499037656 }
  ,{ 0.0004176299431532064, 0.02672831636180521 }
  ,{ -0.01379527033462789, -0.4414486507080926 }
} }
,{ {
  { -5.577155569193423e-08,-7.138759128567582e-06 }
  ,{ -4.459338467442303e-06,-0.0004280964928744611 }
  ,{ 0.0003923350293816243, 0.02510944188042395 }
  ,{ -0.01217428430645078, -0.3895770978064249 }
} }
,{ {
  { -3.605928292969135e-08,-4.615588215000493e-06 }
  ,{ -4.826028411989542e-06,-0.000463298727550996 }
  ,{ 0.0003643606551067552, 0.02331908192683234 }
  ,{ -0.01066016019099486, -0.3411251261118356 }
} }
,{ {
  { -1.781470707992619e-08,-2.280282506230552e-06 }
  ,{ -5.040602043929432e-06,-0.0004838977962172255 }
  ,{ 0.0003346512962838997, 0.02141768296216958 }
  ,{ -0.009261708063277772, -0.2963746580248887 }
} }
,{ {
  { -1.506798321843929e-09,-1.928701851960229e-07 }
  ,{ -5.116754768286062e-06,-0.0004912084577554619 }
  ,{ 0.0003040813783947048, 0.0194612082172611 }
  ,{ -0.007984091541769101, -0.2554909293366112 }
} }
,{ {
  { 1.254240539445633e-08,1.60542789049041e-06 }
  ,{ -5.071346219164424e-06,-0.0004868492370397846 }
  ,{ 0.0002734327802100555, 0.01749969793344355 }
  ,{ -0.006829155307778654, -0.2185329698489169 }
} }
,{ {
  { 2.415382265476675e-08,3.091689299810144e-06 }
  ,{ -4.923236176470265e-06,-0.0004726306729411454 }
  ,{ 0.0002433793645195896, 0.01557627932925373 }
  ,{ -0.00579582856353525,  -0.185466514033128 }
} }
,{ {
  { 3.327970129438388e-08,4.259801765681137e-06 }
  ,{ -4.692183812243332e-06,-0.0004504496459753599 }
```

```
  ,{ 0.0002144783492816111, 0.01372661435402311 }
  ,{ -0.004880576558929733, -0.1561784498857515 }
} }
,{ {
  { 3.998619096881058e-08,5.118232444007755e-06 }
  ,{ -4.397864199620254e-06,-0.0004221949631635444 }
  ,{ 0.0001871679663079738, 0.01197874984371032 }
  ,{ -0.004077873823019379, -0.1304919623366201 }
} }
,{ {
  { 4.443210814183816e-08,5.687309842155285e-06 }
  ,{ -4.059040540924509e-06,-0.0003896678919287529 }
  ,{ 0.0001617705765833013, 0.01035331690133128 }
  ,{ -0.003380675535016474, -0.1081816171205272 }
} }
,{ {
  { 4.684585000529841e-08,5.996268800678196e-06 }
  ,{ -3.692914687568888e-06,-0.0003545198100066132 }
  ,{ 0.0001385002284466404,0.008864014620584983 }
  ,{ -0.002780867190684069,-0.08898775010189021 }
} }
,{ {
  { 4.750223096564192e-08,6.080285563602166e-06 }
  ,{ -3.314663535346774e-06,-0.0003182076993932903 }
  ,{ 0.0001174735554921313,0.007518307551496402 }
  ,{ -0.002269676983939308,-0.07262966348605786 }
} }
,{ {
  { 4.670072464852188e-08, 5.9776927550108e-06 }
  ,{ -2.937155806094876e-06,-0.0002819669573851081 }
  ,{ 9.872290650570907e-05,0.006318266016365381 }
  ,{ -0.001838039771308927,-0.05881727268188565 }
} }
,{ {
  { 4.474623841270917e-08,5.727518516826773e-06 }
  ,{ -2.570833263466638e-06,-0.0002467999932927972 }
  ,{ 8.22106662144394e-05,0.005261482637724121 }
  ,{ -0.00147690580564427,-0.04726098578061663 }
} }
,{ {
  { 4.193316738169349e-08,5.367445424856766e-06 }
  ,{ -2.223732912047361e-06,-0.0002134783595565467 }
  ,{ 6.784384611408351e-05,0.004342006151301345 }
  ,{ -0.001177491364418304,-0.03767972366138573 }
} }
,{ {
```

```
  { 3.853310716189559e-08,4.932237716722635e-06 }
  ,{ -1.901622239253096e-06,-0.0001825557349682972 }
  ,{ 5.548818102150092e-05,0.003551243585376059 }
  ,{ -0.0009314717770673435,-0.02980709686615499 }
} }
,{ {
  { 3.478627717320908e-08,4.452643478170763e-06 }
  ,{ -1.608217862833661e-06,-0.0001543889148320314 }
  ,{ 4.498114169517277e-05,0.002878793068491057 }
  ,{ -0.000731120067225914,-0.02339584215122925 }
} }
,{ {
  { 3.089643929408761e-08,3.954744229643214e-06 }
  ,{ -1.345458635910959e-06,-0.000129164029047452 }
  ,{ 3.614345122621364e-05,0.002313180878477673 }
  ,{ -0.0005693964281980402,-0.01822068570233729 }
} }
,{ {
  { 2.702889954915076e-08,3.459699142291297e-06 }
  ,{ -1.113806810496106e-06,-0.0001069254538076262 }
  ,{ 2.878886012546207e-05,0.001842487048029572 }
  ,{ -0.0004399950596155604,-0.01407984190769793 }
} }
,{ {
  { 2.331105572966068e-08,2.983815133396567e-06 }
  ,{ -9.125546956961835e-07,-8.760525078683361e-05 }
  ,{ 2.273208266980214e-05,0.001454853290867337 }
  ,{ -0.0003373555705483165,-0.01079537825754613 }
} }
,{ {
  { 1.983489619315291e-08,2.538866712723572e-06 }
  ,{ -7.40118807871303e-07,-7.105140555564509e-05 }
  ,{ 1.779491911631872e-05,0.001138874823444398 }
  ,{ -0.0002566462908318581,-0.00821268130661946 }
} }
,{ {
  { 1.66608547873407e-08,2.13258941277961e-06 }
  ,{ -5.943082966062757e-07,-5.705359647420247e-05 }
  ,{ 1.381068205132086e-05,0.0008838836512845352 }
  ,{ -0.0001937265370264521,-0.006199249184846466 }
} }
,{ {
  { 1.382247111915601e-08,1.769276303251969e-06 }
  ,{ -4.725590338093643e-07,-4.536566724569897e-05 }
  ,{ 1.062711036208305e-05,0.0006801350631733152 }
  ,{ -0.000145094266684409,-0.004643016533901087 }
```

```
} }
,{ {
  { 1.133138043388216e-08,1.450416695536916e-06 }
  ,{ -3.721288720513465e-07,-3.572437171692926e-05 }
  ,{ 8.107993188612562e-06,0.000518911564071204 }
  ,{ -0.0001078247346620797,-0.00345039150918654 9 }
} }
,{ {
  { 9.182249868241103e-09,1.175327983134861e-06 }
  ,{ -2.902530201337116e-07,-2.786428993283631e-05 }
  ,{ 6.133742295451233e-06,0.0003925595069088789 }
  ,{ -7.950483672849648e-05,-0.002544154775311887 }
} }
,{ {
  { 7.357376121771672e-09,9.41744143586774e-07 }
  ,{ -2.242611569820876e-07,-2.152907107028042e-05 }
  ,{ 4.601149006582652e-06,0.0002944735364212897 }
  ,{ -5.816687120992353e-05,-0.001861339878717553 }
} }
,{ {
  { 5.830754319011006e-09,7.463365528334088e-07 }
  ,{ -1.716598017483407e-07,-1.647934096784071e-05 }
  ,{ 3.422545861207931e-06,0.0002190429351173076 }
  ,{ -4.222453301828047e-05,-0.001351185056584975 }
} }
,{ {
  { 4.571512315677453e-09,5.85153576406714e-07 }
  ,{ -1.301846350963824e-07,-1.249772496925271e-05 }
  ,{ 2.524568002693763e-06,0.0001615723521724008 }
  ,{ -3.041312172389421e-05,-0.0009732198951646148 }
} }
,{ {
  { 3.546674836120153e-09,4.539743790233795e-07 }
  ,{ -9.782802374705724e-08,-9.391490279717495e-06 }
  ,{ 1.846679051040788e-06,0.0001181874592666104 }
  ,{ -2.173522470186589e-05,-0.0006955271904597086 }
} }
,{ {
  { 2.723275825780344e-09,3.48579305699884e-07 }
  ,{ -7.284706052621752e-08,-6.993317810516882e-06 }
  ,{ 1.339594192283002e-06,8.573402830611214e-05 }
  ,{ -1.541254130223323e-05,-0.0004932013216714634 }
} }
,{ {
  { 2.069875334693146e-09,2.649440428407226e-07 }
  ,{ -5.375712506594504e-08,-5.160684006330723e-06 }
```

```
  ,{ 9.637020384530377e-07,6.167693046099441e-05 }
  ,{ -1.084404604250008e-05,-0.0003470094733600024 }
} }
,{ {
  { 1.557560545084242e-09,1.99367749770783e-07 }
  ,{ -3.931541933447623e-08,-3.774280256109718e-06 }
  ,{ 6.875582939894272e-07,4.400373081532334e-05 }
  ,{ -7.570340751290444e-06,-0.0002422509040412942 }
} }
,{ {
  { 1.160521497792107e-09,1.485467517173897e-07 }
  ,{ -2.849824560521101e-08,-2.735831578100257e-06 }
  ,{ 4.864995334541183e-07,3.113597014106357e-05 }
  ,{ -5.243804289419642e-06,-0.0001678017372614286 }
} }
,{ {
  { 8.562933419045533e-10,1.096055477637828e-07 }
  ,{ -2.04750605380212e-08,-1.965605811650035e-06 }
  ,{ 3.41404983959747e-07,2.184991897342381e-05 }
  ,{ -3.603997550434695e-06,-0.0001153279216139102 }
} }
,{ {
  { 6.25751255928244e-10,8.009616075881524e-08 }
  ,{ -1.458162415139377e-08,-1.399835918533802e-06 }
  ,{  2.3761818240736e-07,1.520756367407104e-05 }
  ,{ -2.457703348469032e-06,-7.864650715100903e-05 }
} }
,{ {
  { 4.529340758609023e-10,5.79755617101955e-08 }
  ,{ -1.029391197359319e-08,-9.882155494649466e-07 }
  ,{ 1.640284771128031e-07,1.04978225352194e-05 }
  ,{ -1.662958424634951e-06,-5.321466958831844e-05 }
} }
,{ {
  { 3.247602445078765e-10,4.15693112970082e-08 }
  ,{ -7.203899323906403e-09,-6.915743350950147e-07 }
  ,{ 1.123040862084225e-07,7.187461517339037e-06 }
  ,{ -1.116452558660085e-06,-3.572648187712273e-05 }
} }
,{ {
  { 2.30686638875616e-10,2.952788977607884e-08 }
  ,{ -4.997870017311665e-09,-4.797955216619198e-07 }
  ,{ 7.626321981856181e-08,4.880846068387956e-06 }
  ,{ -7.43714246992367e-07,-2.379885590375575e-05 }
} }
,{ {
```

```
  { 1.62348343465352e-10,2.078058796356506e-08 }
  ,{ -3.437547837088295e-09,-3.300045923604763e-07 }
  ,{ 5.136699602782352e-08,3.287487745780705e-06 }
  ,{ -4.915626419109025e-07,-1.573000454114888e-05 }
} }
,{ {
  { 1.132062933665611e-10,1.449040555091982e-08 }
  ,{ -2.344090013092694e-09,-2.250326412568986e-07 }
  ,{ 3.43169347778733e-08,2.196283825783891e-06 }
  ,{ -3.223729352241681e-07,-1.031593392717338e-05 }
} }
,{ {
  { 7.822032766619364e-11,1.001220194127279e-08 }
  ,{ -1.5848050842967e-09,-1.521412880924832e-07 }
  ,{ 2.274016527990733e-08,1.455370577914069e-06 }
  ,{ -2.097708834108638e-07,-6.71266826914764e-06 }
} }
,{ {
  { 5.355770518520448e-11,6.855386263706173e-09 }
  ,{ -1.062348163068451e-09,-1.019854236545713e-07 }
  ,{ 1.494668127269781e-08, 9.5658760145266e-07 }
  ,{ -1.354374489379326e-07,-4.333998366013844e-06 }
} }
,{ {
  { 3.634147481032868e-11,4.651708775722071e-09 }
  ,{ -7.060897746027254e-10,-6.778461836186165e-08 }
  ,{ 9.74466484193354e-09,6.236585498837466e-07 }
  ,{ -8.676393515408498e-08,-2.776445924930719e-06 }
} }
,{ {
  { 2.443906800145819e-11,3.128200704186648e-09 }
  ,{ -4.653362120425529e-10,-4.467227635608508e-08 }
  ,{ 6.301801322850928e-09,4.033152846624594e-07 }
  ,{ -5.515014134094942e-08,-1.764804522910381e-06 }
} }
,{ {
  { 1.628883049763073e-11,2.084970303696734e-09 }
  ,{ -3.04087667299206e-10,-2.919241606072378e-08 }
  ,{ 4.042431109848616e-09,2.587155910303114e-07 }
  ,{ -3.478251051578401e-08,-1.113040336505088e-06 }
} }
,{ {
  { 1.076065172092256e-11,1.377363420278088e-09 }
  ,{ -1.970454032946794e-10,-1.891635871628923e-08 }
  ,{ 2.572200970727208e-09,1.646208621265413e-07 }
  ,{ -2.176617521777173e-08,-6.965176069686953e-07 }
```

```
} }
,{ {
  { 7.046112666236051e-12,9.019024212782145e-10 }
  ,{ -1.266132861125826e-10,-1.215487546680793e-08 }
  ,{ 1.623512136833541e-09,1.039047767573466e-07 }
  ,{ -1.351481829664788e-08,-4.324741854927323e-07 }
} }
,{ {
  { 4.573415576263572e-12,5.853971937617372e-10 }
  ,{ -8.067651102423245e-11,-7.744945058326315e-09 }
  ,{ 1.016478927962931e-09,6.505465138962761e-08 }
  ,{ -8.326168379349991e-09,-2.664373881391997e-07 }
} }
,{ {
  { 2.942581325108452e-12,3.766504096138819e-10 }
  ,{ -5.097751607967133e-11,-4.893841543648448e-09 }
  ,{ 6.313018521581508e-10,4.040331853812165e-08 }
  ,{ -5.08963981633602e-09,-1.628684741227526e-07 }
} }
} ;
const double  ErfEvaluator::IntegrationConstants[ErfEvaluator::k_Slices]
= {
  0.9647496350557387
 ,0.8945235826411205
 ,0.8251151582995999
 ,0.7570483478113061
 ,0.6908163429125882
 ,0.6268708869645903
 ,0.5656131765729484
 ,0.5073866186839697
 ,0.452471647684221
 ,0.4010827047956272
 ,0.3533673790720676
 ,0.3094076118069897
 ,0.2692227796948113
 ,0.2327744011161023
 ,0.1999721575621347
 ,0.1706808901260279
 ,0.1447282193300569
 ,0.1219124441277339
 ,0.1020104003572524
 ,0.0847849970053737
 ,0.06999219660381942
 ,0.05738725995438308
 ,0.04673013130128872
 ,0.03778989454781171
```

```
 ,0.03034828122884282
 ,0.02420225448230857
 ,0.01916572874603583
 ,0.01507051162193889
 ,0.01176657225069364
 ,0.00912175014762694
 ,0.007021020716142793
 ,0.005365429817903765
 ,0.004070801226287904
 ,0.003066308931668393
 ,0.002292992435297803
 ,0.001702278540127081
 ,0.001254558696270697
 ,0.0009178574353023258
 ,0.0006666153543141322
 ,0.000480599801055098
 ,0.0003439479971021772
 ,0.0002443408016824203
 ,0.0001723005396039184
 ,0.0001206030943084474
 ,8.379256147784313e-05
 ,5.778591550183871e-05
 ,3.955511222290099e-05
 ,2.687460931683381e-05
 ,1.81232329484865e-05
 ,1.213049270853172e-05
 ,8.058717921777377e-06
 ,5.313661691920574e-06
 ,3.477428056922396e-06
 ,2.258680312250875e-06
 ,1.456062300565673e-06
 ,9.316013422969179e-07
 ,5.915640033304468e-07
 ,3.728134139618971e-07
 ,2.33182677406109e-07
 ,1.447481221205399e-07
 ,8.91740616342884e-08
 ,5.452181280340989e-08
 ,3.308304223623579e-08
 ,1.99223879235018e-08
} ;
```

0.3.6 Logarithm

```
const double Math::hlogTable[16] = {
  0.0 // ::log(1.0) ,
```

```
  , 0.06062462181643484 // ::log(1.0 + 1.0/16.0),
  , 0.11778303565638346 // ::log(1.0 + 2.0/16.0),
  , 0.17185025692665923 // ::log(1.0 + 3.0/16.0),
  , 0.22314355131420976 // ::log(1.0 + 4.0/16.0),
  , 0.27193371548364176 // ::log(1.0 + 5.0/16.0),
  , 0.31845373111853459 // ::log(1.0 + 6.0/16.0),
  , 0.36290549368936847 // ::log(1.0 + 7.0/16.0),
  , 0.40546510810816438 // ::log(1.0 + 8.0/16.0),
  , 0.44628710262841953 // ::log(1.0 + 9.0/16.0),
  , 0.48550781578170082 // ::log(1.0 + 10.0/16.0),
  , 0.52324814376454787 // ::log(1.0 + 11.0/16.0),
  , 0.55961578793542266 // ::log(1.0 + 12.0/16.0),
  , 0.59470710774669278 // ::log(1.0 + 13.0/16.0),
  , 0.62860865942237409 // ::log(1.0 + 14.0/16.0),
  , 0.66139848224536502 // ::log(1.0 + 15.0/16.0)
} ;
const double Math::hlogComp[16] = {
  1.0 / ( 1.0 + 0.0/16.0 ) ,
  1.0 / ( 1.0 + 1.0/16.0 ) ,
  1.0 / ( 1.0 + 2.0/16.0 ) ,
  1.0 / ( 1.0 + 3.0/16.0 ) ,
  1.0 / ( 1.0 + 4.0/16.0 ) ,
  1.0 / ( 1.0 + 5.0/16.0 ) ,
  1.0 / ( 1.0 + 6.0/16.0 ) ,
  1.0 / ( 1.0 + 7.0/16.0 ) ,
  1.0 / ( 1.0 + 8.0/16.0 ) ,
  1.0 / ( 1.0 + 9.0/16.0 ) ,
  1.0 / ( 1.0 + 10.0/16.0 ) ,
  1.0 / ( 1.0 + 11.0/16.0 ) ,
  1.0 / ( 1.0 + 12.0/16.0 ) ,
  1.0 / ( 1.0 + 13.0/16.0 ) ,
  1.0 / ( 1.0 + 14.0/16.0 ) ,
  1.0 / ( 1.0 + 15.0/16.0 )
} ;
 // Log base e, along similar lines
 // (actually computes log(abs(x)), gives a large negative number for
 // f(0), gives a large positive
 //  number if fed Inf or Nan)
 static inline double hlog(double x)
 {
   const DoubleMap m(x) ;
   int exponent = m.Exponent() ;
   unsigned int sig_hi = m.SignificandHiBits() ;
   DoubleMap m1(0,0,sig_hi, m.SignificandLoBits()) ;
   unsigned int tableIndex = sig_hi >> 16 ;
   double xx=m1.GetValue()*hlogComp[tableIndex] - 1.0 ;
```

```
  // There should be scope for shortening this polynomial
  // (1) rescale for 'xx' to be in (-1/32, 1/32) rather than (0,1/16)
  // (2) have a larger table, and take more bits to index it
  // (actually, not sure this is large enough for double precision)
  double p10 =         (-1.0/10.0) ;
  double p9  = p10 * xx + (  1.0 / 9.0 ) ;
  double p8  = p9 * xx + ( -1.0 / 8.0 ) ;
  double p7  = p8 * xx + (  1.0 / 7.0 ) ;
  double p6  = p7 * xx + ( -1.0 / 6.0 ) ;
  double p5  = p6 * xx + (  1.0 / 5.0 ) ;
  double p4  = p5 * xx + ( -1.0 / 4.0 ) ;
  double p3  = p4 * xx + (  1.0 / 3.0 ) ;
  double p2  = p3 * xx + ( -1.0 / 2.0 ) ;
  double p1  = p2 * xx + (  1.0 )  ;
  double p0  = p1 * xx ;
  double result= exponent*M_LN2 + hlogTable[tableIndex] + p0 ;
/*
 * BegLogLine(1)
 *   << "hlog x=" << x
 *   << " exponent=" << exponent
 *   << " tableIndex=" << tableIndex
 *   << " xx=" << xx
 *   << " p0=" << p0
 *   << " result=" << result
 *   << EndLogLine ;
 */
  return result ;
 }
```

0.3.7 Exponential

```
#ifdef DBL_MAX
const double Math::Infinity = DBL_MAX * 2.0 ;  // Intended to overflow;
                        // will be used as result of hexp(x) for x > 709
#else
#if defined(PK_BGL)
// BG/L compiler doesn't support HUGE_VAL quite the way we want ...
const double Math::Infinity = (1e200*1e200) ; // Intended to overflow
#else
const double Math::Infinity = HUGE_VAL * HUGE_VAL ;
#endif
#endif
const double ExpK1 = M_LN2/16.0 ;
const double ExpK2 = M_LN2/256.0 ;
const double Math::ExpTable1[16] = {
//      ::exp( 0.0*ExpK1) ,       ::exp( 1.0*ExpK1) ,
```

```
//      ::exp( 2.0*ExpK1) ,       ::exp( 3.0*ExpK1) ,
        1.00000000000000000000 ,  1.04427378242741383881 ,
        1.09050773266525765605 ,  1.13878863475669164875
//      ::exp( 4.0*ExpK1) ,       ::exp( 5.0*ExpK1) ,
//      ::exp( 6.0*ExpK1) ,       ::exp( 7.0*ExpK1) ,
     ,  1.18920711500272105982 ,  1.24185781207348403959 ,
        1.29683955465100965466 ,  1.35425554693689271456
//      ::exp( 8.0*ExpK1) ,       ::exp( 9.0*ExpK1) ,
//      ::exp(10.0*ExpK1) ,       ::exp(11.0*ExpK1) ,
     ,  1.41421356237309503240 ,  1.47682614593949929212 ,
        1.54221082540794080126 ,  1.61049033194925428250
//      ::exp(12.0*ExpK1) ,       ::exp(13.0*ExpK1) ,
//      ::exp(14.0*ExpK1) ,       ::exp(15.0*ExpK1) ,
     ,  1.68179283050742905681 ,  1.75625216037329945002 ,
        1.83400808640934242627 ,  1.91520656139714725223
   } ;
const double Math::ExpTable2[16] = {
//      ::exp( 0.0*ExpK2) ,       ::exp( 1.0*ExpK2) ,
//      ::exp( 2.0*ExpK2) ,       ::exp( 3.0*ExpK2) ,
        1.00000000000000000000 ,  1.00271127505020248534 ,
        1.00542990111280282117 ,  1.00815589811841751551
//      ::exp( 4.0*ExpK2) ,       ::exp( 5.0*ExpK2) ,
//      ::exp( 6.0*ExpK2) ,       ::exp( 7.0*ExpK2) ,
     ,  1.01088928605170045965 ,  1.01363008495148943838 ,
        1.01637831491095303739 ,  1.01913399607773794904
//      ::exp( 8.0*ExpK2) ,       ::exp( 9.0*ExpK2) ,
//      ::exp(10.0*ExpK2) ,       ::exp(11.0*ExpK2) ,
     ,  1.02189714865411667749 ,  1.02466779289713564431 ,
        1.02744594911876369561 ,  1.03023163768604101185
//      ::exp(12.0*ExpK2) ,       ::exp(13.0*ExpK2) ,
//      ::exp(14.0*ExpK2) ,       ::exp(15.0*ExpK2) ,
     ,  1.03302487902122842138 ,  1.03582569360195711881 ,
        1.03863410196137878930 ,  1.04145012468831613985
   } ;
  /*
   * Branchless exp(x), with a view to vectorising on Double Hummer
   */
  static inline double hexp(double x)
  {
    const double tp10 = 1024.0 ;
    const double tp20 = tp10*tp10 ;
    const double tp40 = tp20*tp20 ;
     // Dividing by ln(2) gives a value such that we can put the
     // integer part into an exponent.
     // Adding (2**44+2**43) aligns and rounds this so that
     // bottom 8 bits can be used for lookup
```

```
  // higher bits can be stuffed into exponent
  // truncated bits (recovered by subtraction) can be fed to
  // power series
 const double x1 = x * ( 1.0 / M_LN2 ) + ( tp40 * ( 16.0 + 8.0 ) ) ;
 const DoubleMap m1(x1) ;
 // Figure the appropriate power of 2 from the significand high bits
 const unsigned int sig_lo = m1.SignificandLoBits() ;
 const DoubleMap m2(0,sig_lo >> 8, 0, 0) ;
 // Recover the number that we will have 'exponentiated' by the bit
 // twiddling
 const double xl2 =  x1 - ( tp40 * ( 16.0 + 8.0 ) ) ;
 // Can range-check xl2 to see if sig_lo put a sensible value in m2
 const double xx4 = xl2 * M_LN2 ;
 // Look up the next several bits (4) in a multiplication table
 const unsigned int tabits =(sig_lo >> 4) & 0x0f ;
 const double x31 = ExpTable1[tabits] ;
 // And the next 4 bits in another table
 const unsigned int tabits2=sig_lo & 0x0f ;
 const double x32 = ExpTable2[tabits2] ;
 const double x3 = x31*x32 ;
 // Figure the remaining part of the original number
 const double z = x - xx4 ;
 // z should be between +- (2**-8); feed in to polynomial for exp(z)
 const double f5 =          1.0/(2.0*3.0*4.0*5.0) ;
 const double f4 = z * f5 + 1.0/(2.0*3.0*4.0) ;
 const double f3 = z * f4 + 1.0/(2.0*3.0) ;
 const double f2 = z * f3 + 1.0/2.0 ;
 const double f1 = z * f2 + 1.0 ;
 const double f0 = z * f1 + 1.0 ;
 const double p0 = f0 * x3 ;
 const double x2=m2.GetValue() ;
 const double r0 = p0 * x2 ;
 // Fixup for out-of-range parameter
 const double resultl = fsel(x+709.0, r0, 0.0) ;
 const double resulth = fsel(x-709.0, Infinity, resultl) ;
 return resulth ;
}
```

0.3.8 'acossin' (inverse sin/cos)

```
template <class T> static inline T asin_small(T x)
   {
      const double ap0 = 1.0           , aq0 = 1.0       ;
      const double ap1 = ap0 * 1.0     , aq1 = aq0 * 2.0 ;
      const double ap2 = ap1 * 3.0     , aq2 = aq1 * 4.0 ;
```

```
   const double ap3 = ap2 * 5.0     , aq3 = aq2 * 6.0 ;
   const double ap4 = ap3 * 7.0     , aq4 = aq3 * 8.0 ;
   const double ap5 = ap4 * 9.0     , aq5 = aq4 * 10.0 ;
   const double ap6 = ap5 * 11.0    , aq6 = aq5 * 12.0 ;
   const double ap7 = ap6 * 13.0    , aq7 = aq6 * 14.0 ;
   const double ap8 = ap7 * 15.0    , aq8 = aq7 * 16.0 ;
   const double ap9 = ap8 * 17.0    , aq9 = aq8 * 18.0 ;
   const double apa = ap9 * 19.0    , aqa = aq9 * 20.0 ;
   const double apb = apa * 21.0    , aqb = aqa * 22.0 ;
   const double apc = apb * 23.0    , aqc = aqb * 24.0 ;
   const double apd = apc * 25.0    , aqd = aqc * 26.0 ;
   const double ape = apd * 27.0    , aqe = aqd * 28.0 ;
   const double apf = ape * 29.0    , aqf = aqe * 30.0 ;
   const double a14 =  apf / ( aqf * 31.0 ) ;
   const double a13 =  ape / ( aqe * 29.0 ) ;
   const double a12 =  apd / ( aqd * 27.0 ) ;
   const double a11 =  apc / ( aqc * 25.0 ) ;      ;
   const double a10 =  apb / ( aqb * 23.0 ) ;
   const double a9  =  apa / ( aqa * 21.0 ) ;
   const double a8  =  ap9 / ( aq9 * 19.0 ) ;
   const double a7  =  ap8 / ( aq8 * 17.0 ) ;
   const double a6  =  ap7 / ( aq7 * 15.0 ) ;
   const double a5  =  ap6 / ( aq6 * 13.0 ) ;
   const double a4  =  ap5 / ( aq5 * 11.0 ) ;
   const double a3  =  ap4 / ( aq4 * 9.0 ) ;
   const double a2  =  ap3 / ( aq3 * 7.0 ) ;
   const double a1  =  ap2 / ( aq2 * 5.0 ) ;
   const double a0  =  ap1 / ( aq1 * 3.0 ) ;
  double b, s, t1, t0;
  s =  a14 + a13 ;
  b = x*x;
  t0 = a14 * b + a13;
  s = b * b;
  t0 = (((((t0*s + a11)*s + a9)*s + a7)*s + a5)*s +a3)*s
    + a1;
  t1 = (((((a12*s + a10)*s + a8)*s + a6)*s + a4)*s + a2);
  return ( x + (x*b)*(a0 + b*(t0 + b*t1)));
}
// Given the sin and cos of an angle, return the angle.
// Returns an angle in (-PI, PI)
inline static double acossin ( double sinang, double cosang )
{
  const double piby8 = M_PI / 8.0 ; // 22.5 degrees, in radians;
  const double pi3by8 = M_PI * ( 3.0 / 8.0 ) ; // 3*22.5 degrees,
                                               // in radians;
  const double pi5by8 = M_PI * ( 5.0 / 8.0 ) ; // 5*22.5 degrees,
```

```
                                                  // in radians;
  const double pi7by8 = M_PI * ( 7.0 / 8.0 ) ; // 7*22.5 degrees,
                                                  // in radians;
  const double cospiby4 = sqrt(2.0) * 0.5 ;
  const double cospiby8 = sqrt((1+cospiby4) * 0.5) ;
  const double sinpiby8 = sqrt(1-cospiby8*cospiby8) ;
  double abscos = fabs(cosang) ; // abscos in (0,1)
  double abssin = fabs(sinang) ; // abssin in (0,1)
  double coslarge = abscos - abssin ;
  // Now we have the sin and cos of an angle between 0 and 90 degrees
  double sincand1 = abssin * cospiby8 - abscos * sinpiby8 ;
                       // sin of an angle in (-22.5,+67.5 degrees)
  double coscand2 = abscos * cospiby8 - abssin * sinpiby8 ;
                       // cos of an angle in (+22.5, 112.5 degrees)
              // which is sin of an angle in (+67.5, -22.5 degrees)
  double trigang = fsel( coslarge , sincand1 , coscand2 );
                      // reduced-range item ready for 'arcsin'
  double ang = asin_small(trigang) ;
  // Now we have an angle which is piecewise-linear related to
  // the wanted one, over the whole circle
  // Compute the multiplier and addend to stitch the angle back
  // together
  // according as which octant we are in; this computation is
  // interleavable
  // since both are branchless
  double km0 = fsel( sinang, 1.0, -1.0 ) ;
  double km1 = fsel( sinang, -1.0, 1.0 ) ;
  double kma = fsel(  coslarge , km0 , km1  ) ;
  double kmb = fsel(  coslarge , km1 , km0 ) ;
  double km  = fsel( cosang , kma , kmb ) ;
  double kaa = fsel( coslarge , piby8, pi3by8 ) ;
  double kab = fsel( coslarge , pi7by8, pi5by8 ) ;
  double ka  = fsel( cosang , kaa , kab ) * km0 ;
  // And stitch the angle back together
  return (ang*km) + ka ;
}
```

0.3.9 Sin and Cos

```
#if !defined(INCLUDE_SINCOS_HPP)
#define INCLUDE_SINCOS_HPP
/*
 * This evaluates 'sin' or 'cos' of an angle, as a single basic block
 * (no branches)
 * so the compiler can schedule the evaluation interleaved with other
 * work.
```

```
 *
 * The angle range is split into
 * (-45, 45), (45, 135), (135, 225), (225, 315) degrees
 * and repeating. According as the range and whether we want
 * 'sin' or 'cos', a
 * suitable even function (sin(x)/x or cos(x)) is evaluated as a
 * Chebyshev polynomial.
 * This is then compensated by multiplication by the
 * appropriate one of
 * +1, +x, -1, or -x, giving the required result
 *
 * As convenional, the argument is taken in radians.
 *
 * The tables for the coefficients of the Chebyshev polynomials
 * are set up separately
 *
 * These tables are good for 8 coefficients, but only the first 7
 * are used to get to
 * double precision
 */
#define A_PI 3.14159265358979323846264338327950288
class TrigConstants
{
   public:
        enum {
                k_Diagnose = 0 ,
                k_ChebSize = 7
        } ;
} ;
#if defined(UNINIT_SINCOSTABLE)
static long double SinCosChebTable [2][TrigConstants::k_ChebSize+1] ;
#else
static const double SinCosChebTable [2][TrigConstants::k_ChebSize+1] = {
{
  -0.0789004058803453350315181615833139406341584620305671897936151525
 ,-0.0391445675270819570174285389007398988869386583741731420316625604
 ,+0.0003045094206789444055815695590884452454366695761311146250488983
 ,-0.0000011235749767964159582142036241454977987906529477530843654213
 ,+0.0000000024140399724137496071057892543727134506131049276895890590
 ,-0.0000000000033916367050375354740011828074494768576077765801943509
 ,+0.0000000000000033580876185142034466964844195879864668971641022093
 ,-0.0000000000000000024689833209931834105086652046149581936814955209
} , {
  -0.2967361725903839745991879698781526954590354296709458772475200482
 ,-0.1464366443908368633207963601399962102709746936143883813474659270
 ,+0.0019214493118146467969071454374523876476540840033801859035372729
```

```
 ,-0.0000099649684898293000686691061850349099578334955892601473193098
 ,+0.0000000275765956071873951864383928564160302546849325446961470666
 ,-0.0000000000473994980816484403744256516400929203293229935948172038
 ,+0.0000000000000554954854148518274108762542315929735058906972782761
 ,-0.0000000000000000470970490651755595726933928753403549933019035260
}
}
;
#endif
#if defined(EXTERN_DK1)
extern double dk1 ;
#else
double dk1 = 1.0 ;
#endif
static const double TrigK0table[4] = { 0.0, 1.0, 0.0, -1.0 } ;
static const double TrigK1table[4] = { 1.0, 0.0, -1.0, 0.0 } ;
class Trig: public TrigConstants
{
        public:
        /*
         * ChebyshevEvaluate is based on 'chebev' which is
         * (c) Numerical Recipes 1988, 1992
         * The compiler unrolls the loop fully, for reasonable 'm'
         */
        template <int m>
        static inline double ChebyshevEvaluate(
#if defined(UNINIT_SINCOSTABLE)
                                long double c[m]
#else
                                const double c[m]
#endif
                               ,double x)
        {
                double d=0.0 ;
                double dd=0.0 ;
                double x2=2.0*x;
                for (int j=m-1;j>=1;j -= 1) {
                        double sv=d;
                        d=x2*d-dd+c[j];
                        dd=sv;
                }
                return x*d-dd+0.5*c[0];
        }
        static inline double NearestInteger(const double x)
        {
           const double two10 = 1024.0 ;
```

```
        const double two50 = two10 * two10 * two10 * two10 * two10 ;
        const double two52 = two50 * 4.0 ;
        const double two51 = two50 * 2.0 ;
        const double offset = two52 + two51 ;
        // Force add and subtract of appropriate constant to drop
        // fractional part
        // .. hide it from the compiler so the optimiser won't
        // reassociate things ..
        const double losebits = (dk1*x) + offset ;
        const double result = (dk1*losebits) - offset ;
        return result ;
     }
     static double Sin(double Angle)
     {
             /* Separate domain into
              * -0.5 .. 0.5 : -45 degree to +45 degrees
              * 0.5 .. 1.5 : 45 degrees to 135 degrees
              * and so on
              */
             double Quadrant = Angle * (2.0/A_PI) ;
             double NearestInt = NearestInteger(Quadrant) ;
             int iQuadrant = NearestInt;
             double Remainder = Quadrant - NearestInt ;
             int iTable = (iQuadrant & 1) ;
             double ChebVariable = Remainder * 2.0 ;
             double f = ChebyshevEvaluate<k_ChebSize>
          (SinCosChebTable[iTable],2.0*ChebVariable*ChebVariable-1.0);
             double k0te = TrigK0table[iQuadrant & 3] ;
             double k1te = TrigK1table[iQuadrant & 3] ;
             double pt = k0te + ChebVariable * k1te ;
             double st = k0te + ( Angle - NearestInt*(A_PI/2.0) )
                                   * k1te ;
             double result = f*pt +  st ;
//           if ( k_Diagnose )
//           {
//                   cout << "Trig::Sin(" << Angle << ") "
//                        << "Remainder=" << Remainder
//                    << " iTable=" << iTable
//                    <<     " iSign=" << iSign
//                        << " ChebVariable=" << ChebVariable
//                        << " f=" << f
//                    << " result=" << result
//                    << endl ;
//           }
             return result ;
     }
```

```
        static double Cos(double Angle)
        {
                /* Separate domain into
                 * -0.5 .. 0.5 : -45 degree to +45 degrees
                 * 0.5 .. 1.5 : 45 degrees to 135 degrees
                 * and so on
                 */
                double Quadrant = Angle * (2.0/A_PI) ;
                double NearestInt = NearestInteger(Quadrant) ;
                int iQuadrant = NearestInt;
                double Remainder = Quadrant - NearestInt ;
                int iTable = (iQuadrant & 1) ;
                double ChebVariable = Remainder * 2.0 ;
                double f = ChebyshevEvaluate<k_ChebSize>
           (SinCosChebTable[1-iTable],2.0*ChebVariable*ChebVariable-1.0);
                double k0te = TrigK0table[iQuadrant & 3] ;
                double k1te = TrigK1table[iQuadrant & 3] ;
                double pt = k1te - ChebVariable * k0te ;
                double st = k1te - (  Angle - NearestInt*(A_PI/2.0) )
                                      *  k0te ;
                double result = f*pt + st ;
//              if ( k_Diagnose )
//              {
//                      cout << "Trig::Cos(" << Angle << ") "
//                           << "Remainder=" << Remainder
//                           << " iTable=" << iTable
//                           << " iSign=" << iSign
//                           << " ChebVariable=" << ChebVariable
//                           << " f=" << f
//                           << " result=" << result
//                           << endl ;
//              }
                return result ;
        }
} ;
#endif
```

0.3.10 Nearest image in periodic volume

```
double dk1 = 1.0 ; // The compiler does not know this is
  // constant, so should not 'optimise' away the rounding below
static inline double NearestInteger(const double x)
{
   const double two10 = 1024.0 ;
   const double two50 = two10 * two10 * two10 * two10 * two10 ;
```

```
   const double two52 = two50 * 4.0 ;
   const double two51 = two50 * 2.0 ;
   const double offset = two52 + two51 ;
   // Force add and subtract of appropriate constant to drop
   // fractional part
   // .. hide it from the compiler so the optimiser won't \
   // reassociate things ..
   const double losebits = (dk1*x) + offset ;
   const double result = (dk1*losebits) - offset ;
   return result ;
}
static inline double NearestImageInFullyPeriodicLine(
  const double a
  , const double b
  , const double Period
  , const double ReciprocalPeriod
  )
{
   const double d = b-a ; // 'Regular' distance between them,
                      // if small enough the result will be 'b'
   const double d_unit = d * ReciprocalPeriod ; // express with respect
                                                 // to unit periodicity,
                            // for -0.5 < d_unit < 0.5 result will be 'b'
   const double d_unit_rounded = NearestInteger( d_unit ) ;
   const double result = b - d_unit_rounded * Period ;
   return result ;
}
static inline double NearestDistanceInFullyPeriodicLine(
  const double a
  , const double b
  , const double Period
  , const double ReciprocalPeriod
  )
{
      const double d = b-a ; // 'Regular' distance between them,
                             // if small enough the result will be 'b'
   const double d_unit = d * ReciprocalPeriod ; // express with respect
                                                // to unit periodicity,
                            // for -0.5 < d_unit < 0.5 result will be 'b'
   const double d_unit_rounded = NearestInteger(d_unit) ;
   const double result = d - d_unit_rounded * Period ;
   return result ;
}
static inline double NearestVectorInFullyPeriodicLine(
  const double a
  , const double b
```

```
  , const double Period
  , const double ReciprocalPeriod
  )
{
   return a-NearestImageInFullyPeriodicLine(a,b,Period,ReciprocalPeriod);
}
inline
void
NearestImageInPeriodicVolume(const XYZ &PositionA, const XYZ &PositionB,
                                   XYZ &Nearest)
  {
  double mX = NearestImageInFullyPeriodicLine(PositionA.mX,
                PositionB.mX, DynVarMgrIF.mDynamicBoxDimensionVector.mX,
                DynVarMgrIF.mDynamicBoxInverseDimensionVector.mX ) ;
  double mY = NearestImageInFullyPeriodicLine(PositionA.mY,
                PositionB.mY, DynVarMgrIF.mDynamicBoxDimensionVector.mY,
                DynVarMgrIF.mDynamicBoxInverseDimensionVector.mY ) ;
  double mZ = NearestImageInFullyPeriodicLine(PositionA.mZ,
                PositionB.mZ, DynVarMgrIF.mDynamicBoxDimensionVector.mZ,
                DynVarMgrIF.mDynamicBoxInverseDimensionVector.mZ ) ;
  Nearest.mX = mX ;
  Nearest.mY = mY ;
  Nearest.mZ = mZ ;
  }
inline
void
NearestVectorInPeriodicVolume(const XYZ &PositionA, const XYZ &PositionB,
                                    XYZ &Nearest)
  {
  double mX = NearestVectorInFullyPeriodicLine(PositionA.mX,
                PositionB.mX, DynVarMgrIF.mDynamicBoxDimensionVector.mX,
                DynVarMgrIF.mDynamicBoxInverseDimensionVector.mX ) ;
  double mY = NearestVectorInFullyPeriodicLine(PositionA.mY,
                PositionB.mY, DynVarMgrIF.mDynamicBoxDimensionVector.mY,
                DynVarMgrIF.mDynamicBoxInverseDimensionVector.mY ) ;
  double mZ = NearestVectorInFullyPeriodicLine(PositionA.mZ,
                PositionB.mZ, DynVarMgrIF.mDynamicBoxDimensionVector.mZ,
                DynVarMgrIF.mDynamicBoxInverseDimensionVector.mZ ) ;
  Nearest.mX = mX ;
  Nearest.mY = mY ;
  Nearest.mZ = mZ ;
  }
inline
double
NearestSquareDistanceInPeriodicVolume(const XYZ &PositionA, const XYZ &PositionB)
  {
```

```
double mX = NearestVectorInFullyPeriodicLine(PositionA.mX,
              PositionB.mX, DynVarMgrIF.mDynamicBoxDimensionVector.mX,
              DynVarMgrIF.mDynamicBoxInverseDimensionVector.mX ) ;
double mY = NearestVectorInFullyPeriodicLine(PositionA.mY,
              PositionB.mY, DynVarMgrIF.mDynamicBoxDimensionVector.mY,
              DynVarMgrIF.mDynamicBoxInverseDimensionVector.mY ) ;
double mZ = NearestVectorInFullyPeriodicLine(PositionA.mZ,
              PositionB.mZ, DynVarMgrIF.mDynamicBoxDimensionVector.mZ,
              DynVarMgrIF.mDynamicBoxInverseDimensionVector.mZ ) ;
return  mX*mX + mY*mY + mZ*mZ ;
}
```

0.3.11 Fragment in range

```
double dk1 = 1.0 ; // The compiler does not know this is
  // constant, so should not 'optimise' away the rounding below
static inline double NearestInteger(const double x)
{
   const double two10 = 1024.0 ;
   const double two50 = two10 * two10 * two10 * two10 * two10 ;
   const double two52 = two50 * 4.0 ;
   const double two51 = two50 * 2.0 ;
   const double offset = two52 + two51 ;
   // Force add and subtract of appropriate constant to drop
   // fractional part
   // .. hide it from the compiler so the optimiser won't
   // reassociate things ..
   const double losebits = (dk1*x) + offset ;
   const double result = (dk1*losebits) - offset ;
   return result ;
}
// note ... 'FracScale' returns unsigned
static inline unsigned int FracScale(double n, double rd)
{
   double t = n*rd ;
   double ti = NearestInteger(t) ;
   double tr = t-ti ;                  // tr should be in (-0.5, 0.5)
   const double two10 = 1024.0 ;
   const double two32 = two10 * two10 * two10 * 4.0 ;
   double tri = tr*two32 ;
   int itri=tri ;
   unsigned int utri = itri ;
   return utri ;              // should be a 32-bit integer
                              // representing the fractional position
 }
```

```
class NeighbourList
{
   public:
   class remainder
   {
      public:
       double e2 ;
       int    a  ;
       int dummy ;
   } ;
   enum {
      k_FragCount = NUMBER_OF_FRAGMENTS
      } ;
   enum {
      k_ScaleShift = 8
      } ;
   const XYZ p ;
   const XYZ k ;
   double x[k_FragCount] ;
   double y[k_FragCount] ;
   double z[k_FragCount] ;
   double e[k_FragCount] ;
   double ex[k_FragCount] ;
   double ey[k_FragCount] ;
   double ez[k_FragCount] ;
   int ix[k_FragCount] ;
   int iy[k_FragCount] ;
   int iz[k_FragCount] ;
   int iex[k_FragCount] ;
   int iey[k_FragCount] ;
   int iez[k_FragCount] ;
   int result[k_FragCount] ;
   NeighbourList(const XYZ& ap, const XYZ& ak): p(ap), k(ak) {
      BegLogLine(PKFXLOG_NSQSOURCEFRAG_SUMMARY)
         << "NeighbourList( p=" << p
         << " k=" << k
         << EndLogLine ;
   } ;
   int ProduceAll(
     const XYZ& aXYZ
     , const XYZ& eXYZ
     , double e0
     , int qstart
     , int qend
   ) {
      int q0 = 0 ;
```

```
    for (int a0=qstart; a0<qend; a0+=1 )
    {
       result[q0] = a0 ;
       q0 += 1 ;
    }
    return q0 ;
   }
int Produce(
  const XYZ& aXYZ
  , const XYZ& eXYZ
  , double e0
  , int qstart
  , int qend
) {
     double x0 = aXYZ.mX ;
     double px = p.mX ;
     double kx = k.mX ;
    /*
     * Slice for slab
     */
    remainder xr[k_FragCount] ;
    remainder yr[k_FragCount] ;
    int q1 = 0 ;
    for (int a0=qstart; a0<qend; a0+=1 )
    {
       double dx = NearestDistanceInFullyPeriodicLine(x0,x[a0],px,kx) ;
       double em = e0 + e[a0] ;
       double ex2 = em*em - dx*dx ;
       xr[q1].e2 = ex2 ;
       xr[q1].a = a0 ;
       double FragmentIndexAdd = fsel(ex2,1.0,0.0) ;
       int IndexAdd = FragmentIndexAdd ;
       q1 += IndexAdd ;
       BegLogLine( PKFXLOG_NSQSOURCEFRAG )
         << "NeighbourList::Produce X"
         << " IndexAdd " << IndexAdd
         << " a0 " << a0
         << " x0 " << x0
         << " x " << x[a0]
         << " dx " << dx
         << " e0 " << e0
         << " ex2 " << ex2
         << " q1 " << q1
         << EndLogLine;
    } /* endfor */
    BegLogLine( PKFXLOG_NSQSOURCEFRAG_SUMMARY1 )
```

```
      << "NeighbourList::Produce X Summary"
      << " q1 " << q1
      << EndLogLine ;
   double y0 = aXYZ.mY ;
     double py = p.mY ;
     double ky = k.mY ;
    /*
     * Slice for cylinder
     */
    int q2 = 0 ;
    for (int b1=0; b1<q1; b1+=1 )
    {
       int a1 = xr[b1].a ;
       double dy = NearestDistanceInFullyPeriodicLine(y0,y[a1],py,ky) ;
       double ey2 = xr[b1].e2 - dy*dy ;
       yr[q2].e2 = ey2 ;
       yr[q2].a = a1 ;
       double FragmentIndexAdd = fsel(ey2,1.0,0.0) ;
       int IndexAdd = FragmentIndexAdd ;
       q2 += IndexAdd ;
       BegLogLine( PKFXLOG_NSQSOURCEFRAG )
         << "NeighbourList::Produce Y"
         << " IndexAdd " << IndexAdd
         << " a1 " << a1
         << " y0 " << y0
         << " y " << y[a1]
         << " dy " << dy
         << " e2 " << xr[b1].e2
         << " ey2 " << ey2
         << " q2 " << q2
         << EndLogLine;
    } /* endfor */
//    BegLogLine( PKFXLOG_NSQSOURCEFRAG_SUMMARY )
//      << "NeighbourList::Produce Y Summary"
//      << " q2 " << q2
//      << EndLogLine ;
   double z0 = aXYZ.mZ ;
     double pz = p.mZ ;
     double kz = k.mZ ;
    /*
     * Slice for sphere
     */
    int q3 = 0 ;
    for (int b2=0; b2<q2; b2+=1 )
    {
       int a2 = yr[b2].a ;
```

```
      double dz = NearestDistanceInFullyPeriodicLine(z0,z[a2],pz,kz) ;
      double ez2 = yr[b2].e2 - dz*dz ;
      result[q3] = a2 ;
      double FragmentIndexAdd = fsel(ez2,1.0,0.0) ;
      int IndexAdd = FragmentIndexAdd ;
      q3 += IndexAdd ;
      BegLogLine( PKFXLOG_NSQSOURCEFRAG )
        << "NeighbourList::Produce Z"
        << " IndexAdd " << IndexAdd
        << " a2 " << a2
        << " z0 " << z0
        << " z " << z[a2]
        << " dz " << dz
        << " e2 " << xr[b2].e2
        << " ez2 " << ez2
        << " q3 " << q3
        << EndLogLine;
   } /* endfor */
   BegLogLine( PKFXLOG_NSQSOURCEFRAG_SUMMARY1 )
     << "NeighbourList::Produce Z Summary"
     << " q3 " << q3
     << EndLogLine ;
   return q3 ;
   } ;
int iProduce(
  const XYZ& aXYZ
  , const XYZ& eXYZ
  , double e0
  , int qstart
  , int qend
) {
   const double tp32 = 1024.0*1024.0*1024.0*4.0 ;
   int aix = FracScale(aXYZ.mX,k.mX) ;
   int aiy = FracScale(aXYZ.mY,k.mY) ;
   int aiz = FracScale(aXYZ.mZ,k.mZ) ;
   int aiex = FracScale(eXYZ.mX,k.mX/(1<<k_ScaleShift))  ;
   int aiey = FracScale(eXYZ.mY,k.mY/(1<<k_ScaleShift))  ;
   int aiez = FracScale(eXYZ.mZ,k.mZ/(1<<k_ScaleShift))  ;
   /*
    * Slice for slab
    */
   int xr[k_FragCount] ;
   int q1 = 0 ;
   for (int a0=qstart; a0<qend; a0+=1 )
   {
      int idx = aix - ix[a0] ;    // Difference in 'x' coordinate,
```

```
                              // scaled on full integer range
   int idxq = idx >> k_ScaleShift ; // Difference in 'x'
             // coordinate, scaled down to keep away from overflows
   int iem = (-aiex) - iex[a0] ; // Max difference for things to be
                              // worth computing, scaled like idxq
   int nexp = iem + idxq  ;
   int nexn = iem - idxq  ;
   unsigned int nex = nexp & nexn ;  // Negative if both of the
                            // above are negative, i.e. in range
   int IndexAdd = nex >> 31 ;
   xr[q1] = a0 ;
   q1 += IndexAdd ;
}
BegLogLine( PKFXLOG_NSQSOURCEFRAG_SUMMARY1 )
  << "NeighbourList::Produce X Summary"
  << " q1 " << q1
  << EndLogLine ;
/*
 * Slice for square prism
 */
int yr[k_FragCount] ;
int q2 = 0 ;
for (int b1=0; b1<q1; b1+=1 )
{
   int a1 = xr[b1] ;
   int idy = aiy - iy[a1] ;
   int idyq = idy >> k_ScaleShift ; // Difference in 'y'
             // coordinate, scaled down to keep away from overflows
   int iem = (-aiey) - iey[a1] ;
   int neyp = iem - idyq ;
   int neyn = iem + idyq ;
   unsigned int ney = neyp & neyn ;
   int IndexAdd = ney >> 31 ;
   yr[q2] = a1 ;
   q2 += IndexAdd ;
} /* endfor */
/*
 * Slice for cube
 */
int zr[k_FragCount] ;
int q3 = 0 ;
for (int b2=0; b2<q2; b2+=1 )
{
   int a2 = yr[b2] ;
   int idz = aiz - iz[a2] ;
   int idzq = idz >> k_ScaleShift ; // Difference in 'z'
```

```
                // coordinate, scaled down to keep away from overflows
      int iem = (-aiez) - iez[a2] ;
      int nezp = iem - idzq ;
      int nezn = iem + idzq ;
      unsigned int nez = nezp & nezn ;
      int IndexAdd = nez >> 31 ;
      zr[q3] = a2 ;
      q3 += IndexAdd ;
   } /* endfor */
   /*
    * Examine cuboid for sphere
    */
   int q4 = 0 ;
   double x0=aXYZ.mX ;
   double y0=aXYZ.mY ;
   double z0=aXYZ.mZ ;
   double px = p.mX ;
   double py = p.mY ;
   double pz = p.mZ ;
   double kx = k.mX ;
   double ky = k.mY ;
   double kz = k.mZ ;
   for (int b3=0; b3<q3 ; b3+=1)
   {
      int a3 = zr[b3] ;
      double dx = NearestDistanceInFullyPeriodicLine(x0,x[a3],px,kx) ;
      double dy = NearestDistanceInFullyPeriodicLine(y0,y[a3],py,ky) ;
      double dz = NearestDistanceInFullyPeriodicLine(z0,z[a3],pz,kz) ;
      double em = e0 + e[a3] ;
      double ex2 = em*em - dx*dx - dy*dy - dz*dz ;
      result[q4] = a3 ;
      double FragmentIndexAdd = fsel(ex2,1.0,0.0) ;
      int IndexAdd = FragmentIndexAdd ;
      q4 += IndexAdd ;
   } /* endfor */
   BegLogLine( PKFXLOG_NSQSOURCEFRAG_SUMMARY1 )
     << "NeighbourList::Produce S Summary"
     << " q4 " << q4
     << EndLogLine ;
   return q4 ;
  } ;
int iProduce_logged(
  const XYZ& aXYZ
  , const XYZ& eXYZ
  , double e0
  , int qstart
```

```
  , int qend
) {
   const double tp32 = 1024.0*1024.0*1024.0*4.0 ;
   int aix = FracScale_logged(aXYZ.mX,k.mX) ;
   int aiy = FracScale_logged(aXYZ.mY,k.mY) ;
   int aiz = FracScale_logged(aXYZ.mZ,k.mZ) ;
   int aiex = FracScale_logged(eXYZ.mX,k.mX/(1<<k_ScaleShift))  ;
   int aiey = FracScale_logged(eXYZ.mY,k.mY/(1<<k_ScaleShift))  ;
   int aiez = FracScale_logged(eXYZ.mZ,k.mZ/(1<<k_ScaleShift))  ;
   BegLogLine( 1 )
     << "iProduce aXYZ=" << aXYZ
     << " k=" << k
     << " eXYZ=" << eXYZ
     << " aix=" << hex << aix
     << " aiy=" << hex << aiy
     << " aiz=" << hex << aiz
     << " aiex=" << hex << aiex
     << " aiey=" << hex << aiey
     << " aiez=" << hex << aiez
     << dec
     << EndLogLine ;
   /*
    * Slice for slab
    */
   int xr[k_FragCount] ;
   int q1 = 0 ;
   for (int a0=qstart; a0<qend; a0+=1 )
   {
      int idx = aix - ix[a0] ;    // Difference in 'x' coordinate,
                                  // scaled on full integer range
      int idxq = idx >> k_ScaleShift ; // Difference in 'x'
           // coordinate, scaled down to keep away from overflows
      int iem = (-aiex) - iex[a0] ; // Max difference for things to be
                                 // worth computing, scaled like idxq
      int nexp = iem + idxq  ;
      int nexn = iem - idxq  ;
      unsigned int nex = nexp & nexn ;  // Negative if both of the
                                  // above are negative, i.e. in range
      int IndexAdd = nex >> 31 ;
      xr[q1] = a0 ;
      q1 += IndexAdd ;
   }
   /*
    * Slice for square prism
    */
   int yr[k_FragCount] ;
```

```
int q2 = 0 ;
for (int b1=0; b1<q1; b1+=1 )
{
   int a1 = xr[b1] ;
   int idy = aiy - iy[a1] ;
   int idyq = idy >> k_ScaleShift ; // Difference in 'y'
             // coordinate, scaled down to keep away from overflows
   int iem = (-aiey) - iey[a1] ;
   int neyp = iem - idyq ;
   int neyn = iem + idyq ;
   unsigned int ney = neyp & neyn ;
   int IndexAdd = ney >> 31 ;
   yr[q2] = a1 ;
   q2 += IndexAdd ;
} /* endfor */
/*
 * Slice for cube
 */
int zr[k_FragCount] ;
int q3 = 0 ;
for (int b2=0; b2<q2; b2+=1 )
{
   int a2 = yr[b2] ;
   int idz = aiz - iz[a2] ;
   int idzq = idz >> k_ScaleShift ; // Difference in 'z'
             // coordinate, scaled down to keep away from overflows
   int iem = (-aiez) - iez[a2] ;
   int nezp = iem - idzq ;
   int nezn = iem + idzq ;
   unsigned int nez = nezp & nezn ;
   int IndexAdd = nez >> 31 ;
   zr[q3] = a2 ;
   q3 += IndexAdd ;
} /* endfor */
/*
 * Examine cuboid for sphere
 */
int q4 = 0 ;
double x0=aXYZ.mX ;
double y0=aXYZ.mY ;
double z0=aXYZ.mZ ;
double px = p.mX ;
double py = p.mY ;
double pz = p.mZ ;
double kx = k.mX ;
double ky = k.mY ;
```

```
    double kz = k.mZ ;
    for (int b3=0; b3<q3 ; b3+=1)
    {
       int a3 = zr[b3] ;
       double dx = NearestDistanceInFullyPeriodicLine(x0,x[a3],px,kx) ;
       double dy = NearestDistanceInFullyPeriodicLine(y0,y[a3],py,ky) ;
       double dz = NearestDistanceInFullyPeriodicLine(z0,z[a3],pz,kz) ;
       double em = e0 + e[a3] ;
       double ex2 = em*em - dx*dx - dy*dy - dz*dz ;
       result[q4] = a3 ;
       double FragmentIndexAdd = fsel(ex2,1.0,0.0) ;
       int IndexAdd = FragmentIndexAdd ;
       q4 += IndexAdd ;
    } /* endfor */
    BegLogLine( 1 )
      << "NeighbourList::Produce S Summary"
      << " q4 " << q4
      << EndLogLine ;
    return q4 ;
   } ;
void SetXYZE(
     int q
   , const XYZ& aXYZ
   , double ae
   , const XYZ& eXYZ
) {
      x[q] = aXYZ.mX ;
      y[q] = aXYZ.mY ;
      z[q] = aXYZ.mZ ;
      ex[q] = eXYZ.mX ;
      ey[q] = eXYZ.mY ;
      ez[q] = eXYZ.mZ ;
      e[q] = ae;
      ix[q] = FracScale(aXYZ.mX,k.mX) ;
      iy[q] = FracScale(aXYZ.mY,k.mY) ;
      iz[q] = FracScale(aXYZ.mZ,k.mZ) ;
      iex[q] = FracScale(eXYZ.mX,k.mX/(1<<k_ScaleShift)) ;
      iey[q] = FracScale(eXYZ.mY,k.mY/(1<<k_ScaleShift)) ;
      iez[q] = FracScale(eXYZ.mZ,k.mZ/(1<<k_ScaleShift)) ;
      BegLogLine( PKFXLOG_NSQSOURCEFRAG_SUMMARY )
        << "NeighbourList::SetXYZE q=" << q
        << " aXYZ=" << aXYZ
        << " ae=" << ae
        << " eXYZ=" << eXYZ
        << EndLogLine ;
    } ;
```

```
    double GetFragmentExtent(int q) const
    {
       return e[q] ;
    } ;
    XYZ GetCorner(int q) const
    {
       XYZ r ;
       r.mX = ex[q] ;
       r.mY = ey[q] ;
       r.mZ = ez[q] ;
       return r ;
    } ;
    XYZ GetFragmentCentre(int q) const
    {
       XYZ r ;
       r.mX = x[q] ;
       r.mY = y[q] ;
       r.mZ = z[q] ;
       return r ;
    } ;
int Get(int q) const { return result[q] ; } ;
} ;
```

Bibliography

[1] Handbook of Mathematical Functions (with Formulas, Graphs, and Mathematical Tables), M. Abramowitz and I.A. Stegun, US Government 1972 , http://dlmf.nist.gov/

[2] Numerical Recipes in C, Press Teukolsky Vetterling and Flannery, Cambridge University Press 1992, http://www.nr.com/