A Fast Exp(-x) Routine

W. D. Cotton, October 4, 2011

***Abstract*—The calculation of the interferometric response to Gaussian componsnts using the direct Fourier transform method involve many evaluations of exp(-x) where x>0. This calculation can dominate the compute time of such models when using the fast sine/cosine evaluation discussed in [1]. This memo describes a technique for fast exp(-x) calculations to moderate accuracy. The technique is a table lookup followed by a single term Taylor's series expansion. An Streaming SIMD Extensions (SSE) implementation further increased the efficiency. A factor of 9 increase in the exp(-x) calculation speed is reported. The precision obtained is more than adequate for the intended applications.**

***Index Terms*—interferometry, performance**

I. Introduction

CAlculating the response of an interferometer to a sky model is a common operation in radio interferometry and for complex sky models can be one of the more computationally expensive operations. One generic type of sky model calculation is the so called Direct Fourier Transform (AKA "DFT") technique wherein the response to each component of a sky model (e.g. CLEAN component) is evaluated for each complex correlation in the data set. Increasing the speed of sine/consne routines is discussed in [1]. For model calculations of the response to Gaussian components many evaluations of exp(-x) are made which dominate the compute time.

The exp routines in standard mathematical libraries have much higher precision that is needed for this radio interferometry application in that they exceed by many orders of magnitude the accuracy of practical phase calibration. Furthermore, the arguments to the exp function involved are negative meaning the function varies from 1 to 0 and is well behaved. Standard library versions of this function are subject to additional constraints such as smoothness and range checking which are unnecessary in the applications discussed here. A moderate relaxation of the precision and other constraints has the potential for significant performance enhancements.

This memo explores such a technique in the Obit package [2] [1].

II. Exp(-x)

For an unresolved Gaussian the argument of the exp function is 0 leading to a value of 1.0. At the other extreme, a strongly resolved Gaussian results in a large negative argument to the exp function leading asymptomatically to zero. Therefore only a finite range of values need to be computed; arguments more negative than some value can be assumed to produce zero. This limited range of arguments allows a lookup table based approach.

National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA, 22903 USA email: bcotton@nrao.edu

[1]http://www.cv.nrao.edu/∼bcotton/Obit.html

While an arbitrary precision can be obtained by a simple table lookup, adequate resolution can require quite large tables. Further improvements in the precision of a table lookup can be obtained using either an interpolation in the table or a series expansion about the tabulated value. A Taylor's series expansion is given by:

$$f(x) \;=\; f(a) \;+\; f'(a)\frac{x-a}{1!} \;+\; f''(a)\frac{(x-a)^2}{2!} + ...$$

Since the derivatives of exp(-x) are the +/-exp(-x) of the same argument, evaluation of such a series is straightforward. The one term expansions for exp x about tabulated value a are:

$$exp(-x) \;=\; exp(a)\;(1-[x-a])$$

III. Implementation

A fast exp(-x) routine, ObitExpCalc, to calculate a exponential was implemented in Obit utility module ObitExp using a 500 element table lookup followed by the single term expansion given above. This routine will initialize the table on the first call. Once initialized, this function should be threadsafe; a single call prior to initializing threading using this routine will make usage threadsafe. The range of allowed values is [0,10], other values being clipped to this range. The exp(-10) is 5.4×10^-5.

A. Vector Implementation

It is possible to go one step further. The overhead of function calls can be reduced by collecting the arguments for which the exp values are desired into an array and doing the calculations in a single function call. A "vector" version of ObitExpCalc is implemented in routine ObitExpVec. Organizing data into vectors also improves the cache hit ratio of the code.

B. SSE

Streaming SIMD Extensions (SSE) is an implementation of SIMD (Single Instruction Multiple Data) architecture for length 4 vectors on Pentium and similar design CPU chipsets. Furthermore, it is supported by the gcc compiler (and likely others) meaning available on all (or nearly all) architectures of interest for radio interferometry. This feature allows coding very fast routines although at the level of assembler and with a Spartan instruction set. This system was designed for fast video game software but has the basics needed for interferometric calculation. A version of ObitSinCosCalc for length 4 vectors was coded using SSE (the most basic version) and implemented in ObitSinCosVec. This code is wrapped in #ifdefs such that if SSE is not available, the non-SSE version will be used. The text of the ObitExp utility package is given in the appendix.

TABLE I
SIN/COS TIMINGS

method	CPU time sec.	ratio
c lib expf	5.84	1.00
ObitExpCalc	1.61	3.63
ObitExpVec c	1.21	4.82
ObitExpVec SSE	0.63	9.26

IV. TESTING

The following give the results of various precision and timing tests.

A. Precision

In order to evaluate the precision of this technique compared to the standard library exp routine, a test program was employed that used 10^8 arguments randomly spaced from 0 to 10 and compared the results of ObitExpCalc and ObitExpVec with the c library expf routine. The average difference in this test was -1.5×10^{-6}, the rms difference was 5.1×10^{-6} and the maximum difference was 5.0×10^{-5}.

B. Exp(-x) Timing

The various possibilities implemented in ObitExp are were compared with the c library exp function using 10^8 arguments randomly chosen from 0 to 10 and run times compared using the Unix time utility. The software was compiled using the -O3 option (highly optimised). Timings of various portions were determined by a comparison of runs with and without that portion. The results are given in Table I where the ratio column gives the speedup ratio compared to the library exp function. These tests were run on a 3 GHz Xenon intel machine; the ObitExpVec SSE routine took about 18 CPU cycles evaluation.

V. DISCUSSION

The cost of "DFT" interferometer model calculations for Gaussian components when using the fast sine/cosine functions described in [1] is strongly dominated by the cost of calculating the exp function. The new vector exp SSE based routine described above appears to be 9 times faster than the c library exp versions.

The precision of this routines appears to be more than adequate and far better than the precision of the calibration. Use of this fast exponential technique can significantly reduce the cost of a common, and expensive, operation in radio interferometry.

APPENDIX

Text of the Obit utility ObitExp.c follows.

REFERENCES

[1] W. D. Cotton, "A Fast Sine/Cosine Routine," *Obit Development Memo Series*, vol. 14, pp. 1–8, 2009.
[2] W. D. Cotton, "Obit: A Development Environment for Astronomical Algorithms," *PASP*, vol. 120, pp. 439–448, 2008.

```
#define OBITEXPTAB  512  /* tabulated points */
/** Is initialized? */
static gboolean isInit = FALSE;
/** Exp lookup table covering minTable to maxTable */
static ofloat exptab[OBITEXPTAB];
/** Arg spacing  in table */
static ofloat delta=0.02;
/** inverse of delta */
static ofloat idelta=50.0;
/** min value tabulated */
static ofloat minTable=1.0e-5;
/**max value tabulated */
static ofloat maxTable=10.0;

/** SSE implementation */
#ifdef HAVE_SSE
#include <xmmintrin.h>

typedef __m128 v4sf;
typedef __m64 v2si;

/* gcc or icc */
# define ALIGN16_BEG
# define ALIGN16_END __attribute__((aligned(16)))

/* Union allowing c interface */
typedef ALIGN16_BEG union {
  float f[4];
  int   i[4];
  v4sf  v;
} ALIGN16_END V4SF;

/* Union allowing c interface */
typedef ALIGN16_BEG union {
  int   i[2];
  v2si  v;
} ALIGN16_END V2SI;

/* Constants */
#define _PS_CONST(Name, Val)                                            \
  static const ALIGN16_BEG float _ps_##Name[4] ALIGN16_END = { Val, Val, Val, Val }
#define _PI32_CONST(Name, Val)                                          \
  static const ALIGN16_BEG int _pi32_##Name[4] ALIGN16_END = { Val, Val, Val, Val }

_PI32_CONST(Obit_NTAB,  512);     /* size of table */
_PS_CONST(Obit_delta, 0.02);      /* table spacing MUST match delta */
_PS_CONST(Obit_idelta, 50.0);     /* 1/table spacing */
_PS_CONST(Obit_minTable, 1.0e-5); /* minimum tabulated value, MUST match minTable */
_PS_CONST(Obit_maxTable, 10.0);   /* maximum tabulated value, MUST match maxTable */
_PS_CONST(Obit_HALF,  0.5);       /* 0.5 */
_PS_CONST(Obit_ONE,   1.0);       /* 1.0 */

#define _OBIT_DELTA     0.02   /* table spacing MUST match delta */
#define _OBIT_IDELTA    50.0   /* 1/table spacing  */
#define _OBIT_MINTABLE  1.0e-5 /* minimum tabulated value, MUST match minTable */
#define _OBIT_MAXTABLE  10.0   /* maximum tabulated value, MUST match maxTable */
#define _OBIT_HALF       0.5   /* 0.5 */
#define _OBIT_ONE        1.0   /* 1.0 */
#define _OBIT_NTAB       512   /* size of table */
```

```
/**
 * Fast vector exp(-arg) using SSE instructions
 * \param arg    argument array
 * \param table  lookup table
 * \param e      [out] array of exp(-arg)
 */
void fast_exp_ps(v4sf arg, float *table, v4sf *e) {
  v4sf cellf, temp, it,  one, exptabl, d;
  V2SI iaddrLo, iaddrHi;

  /* Clip to range */
  temp  = _mm_set_ps1 (_OBIT_MINTABLE);
  arg   = _mm_max_ps (arg, temp);      /* Lower bound */
  temp  = _mm_set_ps1 (_OBIT_MAXTABLE);
  arg   = _mm_min_ps (arg, temp);      /* Upper bound */

  /* get arg in cells */
  cellf = _mm_set_ps1 (_OBIT_MINTABLE);  /* Min table value */
  d     = _mm_sub_ps(arg, cellf);        /* arg-minTable */
  temp  = _mm_set_ps1 (_OBIT_IDELTA);    /* 1/table spacing */
  cellf = _mm_mul_ps(d, temp);           /* (arg-minTable)/table spacing */
  temp  = _mm_set_ps1 (_OBIT_HALF);      /* 0.5 */
  cellf = _mm_add_ps(cellf, temp);
  iaddrLo.v = _mm_cvttps_pi32 (cellf);    /* Round lower half */
  temp      = _mm_movehl_ps (cellf,cellf);/* swap */
  iaddrHi.v = _mm_cvttps_pi32 (temp);     /* Round upper half */

  /* Fetch tabulated values */
  exptabl   = _mm_setr_ps(table[iaddrLo.i[0]],table[iaddrLo.i[1]],
  table[iaddrHi.i[0]],table[iaddrHi.i[1]]);

  /* Get difference in arg from tabulated points */
  temp   = _mm_cvtpi32_ps (temp, iaddrHi.v);      /* float upper values */
  it     = _mm_movelh_ps (temp,temp);           /* swap */
  it     = _mm_cvtpi32_ps (it, iaddrLo.v);      /* float lower values */
  /* it now has the floated, truncated cells */
  temp  = _mm_set_ps1 (_OBIT_DELTA);            /* table spacing */
  temp  = _mm_mul_ps(it, temp);                 /* cell*delta */
  d     = _mm_sub_ps(d, temp);                  /* d = arg-cell*delta-minTable */

  /* One term Taylor's series */
  one   = _mm_set_ps1 (1.0);                   /* ones */
  d     = _mm_sub_ps(one, d);                  /* 1-d */
  *e    = _mm_mul_ps(d, exptabl);              /* table[cell]*(1.0-d) */

  _mm_empty();  /* wait for operations to finish */
  return ;
} /* end fast_exp_ps */

#endif  /* HAVE_SSE */
```

```
/**
 * Initialization
 */
 void ObitExpInit(void)
{
  olong i, cell;
  ofloat arg;

  isInit = TRUE;  /* Now initialized */

  for (i=0; i<OBITEXPTAB-1; i++) {
    arg = minTable + delta * i;
    exptab[i] = exp(-arg);
  }
  exptab[OBITEXPTAB-1] = 0.0;  /* Higher values */

  /* Zero cell for maxTable */
  cell = (olong)((maxTable-minTable)*idelta + 0.5);
  exptab[cell] = 0.0;
} /* end ObitSinCosInit */

/**
 * Calculate exp of -arg
 * arg<minTable => 1.0
 * arg>maxTable => 0.0
 * Lookup table initialized on first call
 * \param arg    argument
 * \return exp(-arg)
*/
ofloat ObitExpCalc(ofloat arg)
{
  olong cell;
  ofloat out, d;

  /* Initialize? */
  if (!isInit) ObitExpInit();

  /* Range test */
  if (arg<minTable) return 1.0;
  if (arg>maxTable) return 0.0;


  /* Cell in lookup table */
  cell = (olong)((arg-minTable)*idelta + 0.5);

  /* Difference from tabulated value */
  d = arg - minTable -cell*delta;

  /* Lookup plus one Taylor term,
     NB, d(exp(-x)/dx = -exp(-x) */
  out = exptab[cell]*(1.0-d);
  return out;
} /* end ObitExpCalc */
```

```
/**
 * Calculate exp(-x) of vector of args uses SSE implementation is available
 * arg<minTable => 1.0
 * arg>maxTable => 0.0
 * Lookup table initialized on first call
 * \param n      Number of elements to process
 * \param argarr array of args
 * \param exparr [out] exp(-arg)
*/
void ObitExpVec(olong n, ofloat *argarr, ofloat *exparr)
{
  olong i, nleft, cell;
  ofloat argt, d;
  /** SSE implementation */
#ifdef HAVE_SSE
  olong ndo;
  V4SF vargt, vex;
#endif /* HAVE_SSE */

  /* Initialize? */
  if (!isInit) ObitExpInit();

  nleft = n;   /* Number left to do */
  i     = 0;   /* None done yet */

 /** SSE implementation */
#ifdef HAVE_SSE
  /* Loop in groups of 4 */
  ndo = nleft - nleft%4;  /* Only full groups of 4 */
  for (i=0; i<ndo; i+=4) {
    vargt.f[0] = *argarr++;
    vargt.f[1] = *argarr++;
    vargt.f[2] = *argarr++;
    vargt.f[3] = *argarr++;
    fast_exp_ps(vargt.v, exptab, &vex.v);
    *exparr++ = vex.f[0];
    *exparr++ = vex.f[1];
    *exparr++ = vex.f[2];
    *exparr++ = vex.f[3];
  } /* end SSE loop */
#endif /* HAVE_SSE */
```

```
  nleft = n-i;  /* How many left? */

 /* Loop doing any elements not done in SSE loop */
  for (i=0; i<nleft; i++) {
    /* arg  */
    argt = (*argarr++);

    /* range check */
    if (argt<minTable) {*exparr++=1.0; continue;}
    if (argt>maxTable) {*exparr++=0.0; continue;}

    /* Cell in lookup table */
    cell = (olong)((argt-minTable)*idelta + 0.5);

    /* Difference from tabulated value */
    d = argt - minTable -cell*delta;

    /* Lookup plus one Taylor term,
       NB, d(exp(-x)/dx = -exp(-x) */
    *exparr++ = exptab[cell]*(1.0-d);
  } /* end loop over vector */
} /* end ObitExpVec */
```