Applied and
Computational
Mathematics
Division

NISTIR 4615

Computing and Applied Mathematics Laboratory

Portable Vectorized Software for Bessel Function Evaluation

Ronald F. Boisvert and Bonita V. Saunders

June 1991

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Gaithersburg, MD 20899

NISTIR 4615

Portable Vectorized Software for Bessel Function Evaluation

Ronald F. Boisvert
Bonita V. Saunders

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Computing and Applied Mathematics Laboratory
Applied and Computational Mathematics Division
Gaithersburg, MD 20899

June 1991

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

Portable Vectorized Software for Bessel Function Evaluation

Ronald F. Boisvert and Bonita V. Saunders*
Computing and Applied Mathematics Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

June 19, 1991

Abstract

A suite of computer programs for the evaluation of Bessel functions and modified Bessel functions of orders zero and one for a vector of real arguments is described. Distinguishing characteristics of these programs are that (a) they are portable across a wide range of machines, and (b) they are vectorized in the case when multiple function evaluations are to be performed. The performance of the new programs are compared with software from the FNLIB collection of Fullerton on which the new software is based.

Keywords: Bessel function, hyperbolic Bessel function, mathematical software, modified Bessel function, order zero and one, portable software, special function, vectorized software.

1 Introduction

Bessel functions of real argument and integer order are among the most commonly occurring special functions of applied mathematics, and most software libraries contain routines for their evaluation. One of the most successful collections of routines for evaluating these and other special functions is the FNLIB package developed by Wayne Fullerton at the Los Alamos National Laboratories in the late 1970s [9]. One of the most important features of FNLIB is its portability. Parameterized by the PORT machine constants [8], FNLIB codes are regularly used on machines from IBM PCs to Cray Y-MPs. Versions of these codes have found their way into several well-known libraries such as the IMSL SFUN/LIBRARY [10], and the SLATEC Common Math Library [2]. They are also available from *netlib* [6].

*Electronic mail: boisvert@cam.nist.gov and saunders@cam.nist.gov, respectively

More recently, increased attention has been paid to the development of algorithms and software which take advantage of vector processors. On such machines, for example, special versions of many standard Fortran math functions are available, so that the compiler can vectorize loops such as the following.

```
      DO 10 I=1,M
         Y(I) = EXP(X(I))
   10 CONTINUE
```

The ability to vectorize such loops is crucial in many applications.

In this paper we describe a set of Fortran-callable subprograms which extend this functionality to the Bessel functions I_0, I_1, J_0, J_1, K_0, K_1, Y_0, and Y_1. This has been done by producing modified versions of the FNLIB routines BESI0, BESI1, BESJ0, BESJ1, BESK0, BESK1, BESY0 and BESY1, as well as their double precision versions. The new routines maintain the portability of FNLIB with the advantage of being vectorizable in cases when multiple function evaluations are required. In Section 2 we review the basic design of the FNLIB routines. In Section 3 we discuss various issues involved in the vectorization of these algorithms. This is followed by a short description of the user interface of our implementation in Section 4. Finally, in Section 5 we describe the testing of the new software, and evaluate its performance on various scalar and vector processors.

2 Design of FNLIB

FNLIB is a substantial collection of software, including more than 200 Fortran subprograms. Double precision versions are available for most codes. Primary design criteria for the development of the package were portability and maintainability. In some cases, other criteria such as speed and accuracy were relaxed slightly in order to maximize the primary criteria [9]. For example, FNLIB routines are rarely accurate to the last bit, although they are almost always accurate to within a factor of 10 times the machine precision. This is certainly sufficient for the vast majority of applications. The fact that these routines have remained popular for more than 10 years, and have been trivially ported to many machines which did not exist when they were initially developed attests to the success of the basic design.

FNLIB is based upon approximation by truncated Chebyshev series expansions [7]. Such approximations have many well-known properties: they are widely applicable, they are nearly best in the minimax sense, the error is easy to estimate, they provide variable accuracy approximations, and they are more stable to evaluate than conventional polynomials. Because of these properties, such approximations have been the basis for many algorithms for the evaluation of special functions.

The basic formula for the construction of the routines in FNLIB, which we paraphrase from [9], follows.

Recipe for Constructing FNLIB Routines

1. *Store Chebyshev coefficients in DATA statements.*
   Coefficients of all required Chebyshev expansions accurate to 16 digits are stored in sin-

gle precision routines, 31 digits in double precision routines. This range ensures sufficient accuracy for most computers.

2. *Initialize.*
   This is done only on the first call of each routine.

   (a) *Calculate legal argument bounds.*
   Illegal argument regions are those where the function is undefined, the result would overflow or underflow, or range reduction cannot be performed accurately enough. The PORT machine constants are used to determine bounds applicable to the current machine.

   (b) *Determine correct number of expansion coefficients.*
   The error committed in truncating a Chebyshev series is bounded by the sum of the absolute values of the discarded coefficients. One can use this to estimate the error committed by using fewer and fewer terms. In this way the length of the series can be selected to match the precision of the machine.

3. *Check input argument for validity.*
   The function argument is compared against bounds computed during initialization. Both fatal errors and warnings are provided. Warnings are issued for underflowed results, for example, while fatal errors are issued in cases where no result can be returned, such as where the result overflows. Errors are issued using the PORT error handler [8].

4. *Compute the approximation.*
   The basic computation is the evaluation of a truncated Chebyshev series. It is not reasonable for a single series to be accurate over the whole argument range. Thus, the argument range is broken up into several subintervals (usually three or four), each with its own series. Once the appropriate subinterval has been determined, the computation proceeds as follows.

   (a) *Preprocess argument.*
   The argument is mapped to the interval [-1,1].

   (b) *Evaluate Chebyshev series.*
   This is done using a three-term linear recurrence due to Clenshaw [3].

   (c) *Postprocess result.*
   The series value is finally corrected for form or interval, if necessary.

3 Vectorization for Multiple Arguments

In this section we describe how routines with the above design may be extended for the case of evaluation at multiple arguments. Three opportunities for improving performance exist: (1) reduction of subprogram call overhead, (2) vectorization of pre- and postprocessing phases, and (3) vectorization of Chebyshev series evaluation. The emphasis on this paper is on (2) and (3).

The vectorization of function evaluation routines for multiple arguments seems quite simple at first — one simply applies the scalar algorithm to the vector of arguments. Unfortunately, this fails because the range of possible input arguments is divided into several

subintervals, each of which is handled differently. Since we cannot assume that the input arguments have been sorted in any way, the core of the algorithm proceeds as follows.

For each argument range:

Gather arguments from this range into a temporary vector.
Compute the approximation for these arguments in vector mode.
Scatter results back into resultant vector.

Clearly, the vector lengths will depend upon the distribution of input arguments, and, in general, will be less than the total number of arguments.

To illustrate this transformation we consider the following linestatement from the FNLIB routine BESI0[1].

```
      IF ((Y .GT. XSML) .AND. (Y .LE. 3.0))
     +   BESI0 = 2.75 + CSEVL(Y*Y/4.5-1.0, BI0CS, NTI0)
```

This line illustrates how one particular argument range is handled. The approximation in this case is simply 2.75 plus the Chebyshev series sum returned by the utility CSEVL. BI0CS is an array of Chebyshev series coefficients and NTI0 is the number of coefficients. The corresponding vector code is more complex.

```
      CALL WGTLE(M,Y,XSML,3.0E0,N,INDX)
      IF (N .GT. 0) THEN
         CALL WGTHR(N,Y,YCMP,INDX)
         DO 20 J=1,N
            TCMP(J) = YCMP(J)**2/4.50E0 - 1.0E0
20       CONTINUE
         CALL WCS(N,TCMP,BI0CS,NTI0,ZCMP,B0,B1,B2)
         DO 30 J=1,N
            ZCMP(J) = 2.750E0 + ZCMP(J)
30       CONTINUE
         CALL WSCTR(N,ZCMP,INDX,F)
      ENDIF
```

Here there are M arguments in the array Y for which the function is to be evaluated with result returned in the array F. WGTLE returns an array INDX of the N indices of elements of the vector Y that are between XSML and 3. These are gathered by the routine WGTHR into YCMP. The 20 loop preprocesses the argument array. WCS evaluates the same Chebyshev series as CSEVL, except for a vector of N arguments stored in the array TCMP. B0, B1, and B2 are work arrays of length M. The 30 loop postprocesses the result of the series evaluation, and then WSCTR puts the result into the appropriate positions of F. The preprocessing and postprocessing phases are trivially vectorizable, while the routines WGTLE, WGTHR, and WSCTR represent utility operations which are also vectorizable.

[1]This statement has been modified slightly from the original code to simplify the presentation.

Next we consider how the evaluation of truncated Chebyshev series is vectorized. First we review how the series is evaluated in the scalar code. Given an argument x and a set of Chebyshev coefficients $c_i, i = 1, \ldots, n$, the following algorithm due to Clenshaw evaluates

$$f(x) = \frac{1}{2}c_1 T_0(x) + \sum_{i=1}^{n-1} c_{i+1} T_i(x).$$

Algorithm 1 : Clenshaw Recurrence — Scalar Version

0. Initialize

$\beta \leftarrow 0$
$\gamma \leftarrow 0$

1. Recurrence for series

for $i = n$ step -1 to 1 do :
$\alpha \leftarrow \gamma$
$\gamma \leftarrow \beta$
$\beta \leftarrow 2x\gamma - \alpha + c_i$

2. Last term; result is in f

$f \leftarrow (\beta - \alpha)/2$

When we have a vector of arguments, x_j, $j = 1, \ldots, m$, it is clear what should be done: x, α, β, γ, and f become vectors and all assignments are loops that run for $j = 1, \ldots, m$. When this is done, however, the recurrence loop has two unnecessary vector copies. These can be eliminated when the loop on i is unrolled to a level of three. This is illustrated in the following vector version of the above algorithm; note that three temporary vectors of length m are required in addition to the input vector x and the output vector f.

Algorithm 2 : Clenshaw Recurrence — Vector Version

0. Initialize

$k \leftarrow n \bmod 3$
$\beta_j \leftarrow 0 \quad$ for $j = 1, \ldots, m$
$\gamma_j \leftarrow 0 \quad$ for $j = 1, \ldots, m$
$f_j \leftarrow 2x_j \quad$ for $j = 1, \ldots, m$

1. Recurrence for series (unrolled)

for $i = n$ step -3 to $1 + k$ do :
$\alpha_j \leftarrow f_j\beta_j - \gamma_j + c_i \quad$ for $j = 1, \ldots, m$
$\gamma_j \leftarrow f_j\alpha_j - \beta_j + c_{i-1} \quad$ for $j = 1, \ldots, m$
$\beta_j \leftarrow f_j\gamma_j - \alpha_j + c_{i-2} \quad$ for $j = 1, \ldots, m$

2. Last term; cleanup for n not divisible by three

Table 1: User-callable VFNLIB subprograms.

VFNLIB		Description	FNLIB	
Single	Double		Single	Double
VI0	DVI0	Evaluates I_0 for a vector of arguments.	BESI0	DBESI0
VI1	DVI1	Evaluates I_1 for a vector of arguments.	BESI1	DBESI1
VJ0	DVJ0	Evaluates J_0 for a vector of arguments.	BESJ0	DBESJ0
VJ1	DVJ1	Evaluates J_1 for a vector of arguments.	BESJ1	DBESJ1
VK0	DVK0	Evaluates K_0 for a vector of arguments.	BESK0	DBESK0
VK1	DVK1	Evaluates K_1 for a vector of arguments.	BESK1	DBESK1
VY0	DVY0	Evaluates Y_0 for a vector of arguments.	BESY0	DBESY0
VY1	DVY1	Evaluates Y_1 for a vector of arguments.	BESY1	DBESY1

if ($k = 0$) then
$f_j \leftarrow (\beta_j - \alpha_j)/2$ for $j = 1, \ldots, m$
elseif ($k = 1$) then
$\alpha_j \leftarrow f_j\beta_j - \gamma_j + c_1$ for $j = 1, \ldots, m$
$f_j \leftarrow (\alpha_j - \gamma_j)/2$ for $j = 1, \ldots, m$
elseif ($k = 2$) then
$\alpha_j \leftarrow f_j\beta_j - \gamma_j + c_2$ for $j = 1, \ldots, m$
$\gamma_j \leftarrow f_j\alpha_j - \beta_j + c_1$ for $j = 1, \ldots, m$
$f_j \leftarrow (\gamma_j - \beta_j)/2$ for $j = 1, \ldots, m$
endif

The loop on i repeatedly utilizes four vectors which can remain in vector registers for the entire computation. Unrolling the loops in this way yields a 20% to 25% improvement in the overall performance of our software on the Cray.

4 The Vector Codes

Using techniques described in the section above, we have produced portable vectorized routines for evaluating the following Bessel functions of real argument and orders zero and one: I_0, I_1, J_0, J_1, K_0, K_1, Y_0, and Y_1 [1]. Since our codes employ the basic algorithms of FNLIB retooled for a vector environment we have called our collection of codes VFNLIB. Although we have produced new user-callable versions of only a small portion of the original FNLIB, the techniques which we have used can be readily applied to many of the remaining FNLIB routines.

VFNLIB includes the 16 user-callable Fortran subprograms which are listed in Table 1. Each has an identical calling sequence.

```
CALL name (M, X, F, WORK, IWORK, INFO)
```

X is an array containing the M arguments for which the function is to be evaluated. The results are returned in the corresponding positions of the array F. WORK and IWORK are workspace arrays. IWORK is of length M, while WORK is of length 7M. INFO is a return code. Both single precision and double precision versions of each function are provided.

We have elected to provide an error return code parameter rather than issue all error messages through an error handler as is done by FNLIB. Error conditions are indicated by a nonzero return code INFO. Fatal errors are indicated by INFO > 0, while warnings have INFO < 0. If any argument leads to a fatal error then no results are returned. Fatal error conditions are: M ≤ 0, function undefined for an argument, a result overflows, an argument so large that accurate argument reduction is not possible. In the last three cases the index of the first offending argument is returned in IWORK(1). A warning is issued when any result underflows; the corresponding function value is set to zero in this case.

We have used a separate procedure to handle errors detected in lower-level routines. An example is that an n-term Chebyshev series is to be evaluated for $n < 1$. Such errors are not caused by improper user input and should never occur during the normal use of these routines. (The exception is when the array WORK is too short and the code overwrites other data.) The only reasonable thing to do in these cases is to issue a fatal error message and halt the program. This is done by calling an very simple error-handling routine WFERR.

A list of all subsidiary routines included in VFNLIB is given in Table 2. All codes are written in standard Fortran 77. A number of common operations such as vector index compression, vector gather and vector scatter are used in VFNLIB. These operations are the most likely to lead to compiler vectorization failures. Although they each vectorize on both the Convex and the Cray Y-MP, they did not vectorize on the Cyber 205, for example. As a result, we have encapsulated these operations into low-level utilities which, if necessary, can be replaced by processor-dependent utilities.

5 Evaluation

5.1 Portability and Accuracy

Since VFNLIB represents a vectorization of a subset of FNLIB, our first goal is to assure that VFNLIB routines return the same results as the corresponding FNLIB routines. A test program that is distributed with the package compares each function with its FNLIB counterpart for approximately 23,000 arguments, verifying that the two differ by no more than five times the machine epsilon. (Relative difference is used for function values greater than one, absolute difference for those less than one.) This code was run on three separate computer systems: Sun SPARCstation 1+ (Sun OS 4.0.3) using Sun Fortran 1.3.1 (options -fast -O3), Convex C-120 (Convex Unix 4.2 release 9.0) using Convex fc version V6.1 (option -O2), Cray Y-MP2/216 (UNICOS 6.0) using cf77. In each case available options were used to check for conformance to ANSI X3.9-1978 FORTRAN.

Some additional testing was done to verify that both sets of routines were indeed producing *correct* function values. Ideally, function routines return results with relative error near the machine precision. This is rarely the case in practice. The accuracy of the VFNLIB

Table 2: Internal VFNLIB subprograms.

VFNLIB		Description	FNLIB	
Single	Double		Single	Double
WI0	DWI0	Evaluates I_0 for a vector of arguments.		
WI1	DWI1	Evaluates I_1 for a vector of arguments.		
WJ0	DWJ0	Evaluates J_0 for a vector of arguments.		
WJ1	DWJ1	Evaluates J_1 for a vector of arguments.		
WK0	DWK0	Evaluates K_0 for a vector of arguments.		
WK1	DWK1	Evaluates K_1 for a vector of arguments.		
WY0	DWY0	Evaluates Y_0 for a vector of arguments.		
WY1	DWY1	Evaluates Y_1 for a vector of arguments.		
WCS	DWCS	Evaluates a given Chebyshev series for a vector of arguments.	CSEVL	DCSEVL
IWCS	IDWCS	Determines number of terms necessary to compute a given Chebyshev series to a given accuracy .	INITS	INITDS
WNGT	DWNGT	Determines if elements of a vector are greater than a given scalar.		
WNLE	DWNLE	Determines if elements of a vector are less than or equal to a given scalar.		
WGT	DWGT	Index compression. Given a vector x_i, and scalars a and b, this constructs an array of indices j for which $x_j > b$.		
WLE	DWLE	Index compression. Given a vector x_i, and scalars a and b, this constructs an array of indices j for which $x_j \leq b$.		
WGTLE	DWGTLE	Index compression. Given a vector x_i, and scalars a and b, this constructs an array of indices j for which $a < x_j \leq b$.		
WGTHR	DWGTHR	Vector gather.		
WSCTR	DWSCTR	Vector scatter.		
WFERR		Processes a fatal error message.	SETERU	
I1MACH	I1MACH	Returns integer valued machine dependent constants. From PORT [8].	I1MACH	I1MACH
R1MACH	D1MACH	Returns real valued machine dependent constants. From PORT [8].	R1MACH	D1MACH

routines was assessed by evaluating each function for four to five thousand arguments on the Cray Y-MP with both FNLIB and VFNLIB and then comparing the results with those computed in extended precision using Mathematica [11]. In most cases the results returned by FNLIB and VFNLIB have relative error less than about 10 times the machine epsilon (1.4×10^{-14}in single precision, 1.0×10^{-28} in double precision). Two notable exceptions to this behavior occur for the functions J_0, J_1, Y_0, and Y_1, which we describe below.

Cancellation at function zeros. These functions are oscillatory, and cancellation errors in the evaluation of the Chebyshev series lead to a loss of relative accuracy in the neighborhood of their zeros (except for $J_1(0) = 0$). In these cases the absolute error remains less than about 10 times the machine epsilon.

Argument reduction for sine and cosine. For large arguments, these functions are computed as $A(x)g(\theta(x))$, where A and θ are given by Chebyshev series, and g is a sine or cosine. Since $\theta(x)$ is of the same size as x, the sine or cosine of a large argument must be computed when x is large. Argument reduction within the sine and cosine procedures can lead to loss of precision for large x.

5.2 Efficiency

The chief motivation for using VFNLIB is the prospect of improved computation rates on vector processors. To assess the advantage of using VFNLIB, we compare the use of VFNLIB routines ("vector codes") with repeated calls of corresponding FNLIB routines ("scalar codes"). We compare the codes on the three systems described in Section 4: Sun SPARCstation 1+, Convex C-120, and Cray Y-MP2. The first is a scalar workstation, the second a vector minisupercomputer, and the third an 2-processor vector supercomputer (although we only use one processor in these tests). We include the Sun to demonstrate the portability of the software, as well as to show that use of VFNLIB need not degrade performance on on scalar processors. All tests were performed in single precision (the Sun and Convex are 32-bit machines while the Cray is a 64-bit machine).

The computation in the innermost loop of the vectorized codes is the evaluation of Chebyshev series using the algorithm of Section 3. In Table 3 we list the computation rate in megaflops for evaluating a single Chebyshev series for 2000 arguments using both scalar and vector codes. The subprograms `CSEVL` and `WCS` were used for this purpose. The theoretical peak rate for the Convex and Cray are 40 and 333 megaflops, respectively, while the corresponding Linpack benchmark runs at 6.5 and 90 megaflops [5]. In light of these figures, the observed rate for vectorized Chebyshev series evaluation on the Convex and Cray, 15 and 182, respectively, are seen to be quite reasonable.

Unfortunately, as we have seen, there is more to the computation of Bessel functions than the evaluation of Chebyshev series. Overhead must be paid for the gathering and scattering of vector elements, as well as for various error checks. It is easy to assess the effect of this overhead on the Cray Y-MP using its hardware performance monitor. Table 4 lists the measured asymptotic computation rate in megaflops for each user-callable function. To obtain this data, we evaluated each function for 19 different sets of 2000 arguments,

Table 3: Asymptotic Performance of Chebyshev Series Evaluation (Megaflops)

Machine	Terms	Scalar	Vector
Sun SPARC 1+	7	1.8	2.9
Convex C-120	7	1.2	15.0
Cray Y-MP2	20	12.7	182.1

Table 4: Asymptotic Performance of Bessel Functions on Cray Y-MP (Megaflops)

Function	I_0	I_1	J_0	J_1	K_0	K_1	Y_0	Y_1
Scalar	9.0	8.7	10.6	10.4	7.8	8.0	9.1	9.0
Vector	148.7	142.4	164.5	160.9	162.8	159.3	172.0	170.0

each with a different argument distribution. Comparing Table 3 and Table 4 we see that computation rates for full Bessel function evaluation decline only 7 to 22 percent over raw Chebyshev series evaluation.

Another measure of performance is speedup, the ratio of scalar to vector times for multiple function evaluation. Actual speedups depend upon the distribution of arguments. We compare the scalar and vector codes for vectors of length 2000 distributed in 19 different ways among three different argument ranges. The test includes cases where all arguments are in a single range as well as cases where only one per cent of the arguments are in a given range. Table 5 lists minimum, average, and maximum speedups observed with the single precision codes on each computer. The speedups for the Convex are in the range of 4.4 to 9.6 while the speedups on the Cray are in the range 13.9 to 23.0. (The entry labelled "Convex C-120 with VECLIB" will be explained shortly.) It is also interesting to note that the vectorized codes run faster than the scalar codes on the Sun in most cases.

The average speedups for the Cray Y-MP given in Table 5 are somewhat better than one might predict from the computation rates given in Table 3. In contrast, the speedups for the Convex are not as large as one might predict. To investigate this further we generated an execution profile of the subroutine VI0 evaluating I_0 for a vector of length 2000 with $x_i = 1$ for all i. The first two columns of Table 6 show the percent time spent by the Cray and the Convex on four different activities. The innermost loop of the computation is Chebyshev series evaluation, which is where more than half the computation occurs on the Cray. However, on the Convex, more than half the time is spent doing index compression; although the compiler does "vectorize" the index compression loops, the resulting code does not perform very well.

In VFNLIB operations such as index compression and gather/scatter have been isolated in low-level utility routines so that they can be easily replaced on systems whose Fortran compilers cannot adequately vectorize them. On the Convex the index compression and gather/scatter loops may be replaced with calls to VECLIB library which has been optimized

Table 5: Asymptotic Speedups (Scalar time / vector time)

Machine		I_0	I_1	J_0	J_1	K_0	K_1	Y_0	Y_1
Sun SPARC 1+	Min	1.0	1.0	1.1	1.1	1.0	1.0	1.2	1.2
	Ave	1.1	1.1	1.2	1.2	1.2	1.2	1.3	1.3
	Max	1.2	1.2	1.4	1.2	1.3	1.4	1.6	1.5
Convex C-120	Min	4.4	5.2	5.5	4.6	7.6	7.3	7.9	7.9
	Ave	5.8	5.9	6.4	5.6	8.0	8.0	8.4	8.5
	Max	6.5	6.2	6.8	6.0	8.3	9.1	9.3	9.6
Convex C-120	Min	6.2	7.5	6.6	5.8	8.2	9.1	8.6	8.0
with VECLIB	Ave	7.9	7.9	7.3	6.6	9.8	9.6	9.2	8.6
	Max	8.8	8.2	7.6	7.0	10.9	10.4	10.3	9.7
Cray Y-MP2	Min	16.2	15.3	14.5	13.9	20.8	20.9	14.8	14.9
	Ave	17.8	16.8	16.0	15.3	21.5	21.6	19.6	19.8
	Max	20.7	19.9	21.5	20.4	22.6	23.1	23.4	23.0

Table 6: Percent Time Spent by VI0 on Various Activities

Activity	Cray	Convex	Convex with VECLIB
Chebyshev series evaluation	57.7	22.8	34.8
Pre- and post- processing	12.1	10.2	15.8
Index compression	20.0	50.2	20.2
Gather/scatter	5.2	9.7	17.9

Table 7: Short Vector Speedups (Scalar time / vector time)

Machine	length	I_0	I_1	J_0	J_1	K_0	K_1	Y_0	Y_1
Sun SPARC 1+	2	0.5	0.5	0.5	0.5	0.6	0.6	0.6	0.6
	20	1.2	1.0	1.0	1.2	1.2	1.2	1.1	1.2
Convex C-120	2	0.2	0.2	0.2	0.2	0.3	0.3	0.2	0.2
	20	1.8	1.8	1.8	1.6	2.2	2.3	2.0	2.2
Cray Y-MP2	2	0.4	0.4	0.4	0.4	0.5	0.5	0.4	0.5
	20	3.6	3.5	3.5	3.4	4.6	4.5	4.0	4.0

Table 8: Short Vector Break-even Point (Vector time = scalar time)

Machine	I_0	I_1	J_0	J_1	K_0	K_1	Y_0	Y_1
Sun SPARC 1+	12	12	14	10	8	8	7	7
Convex C-120	10	10	10	11	8	7	9	8
Cray Y-MP2	5	5	5	5	4	4	5	5

for use on the Convex [4]. When this is done the entries in the third column of Table 6 are obtained. This is is a more reasonable distribution, although the gather/scatter operations now occupy a larger percentage of time than on the Cray. Using the modified version of VFNLIB's utilities on the Convex leads to the the improved scalar/vector speedups labelled "Convex C-120 with VECLIB" in Table 5.

The tests that we have performed thus far assess the performance of the codes for very long vectors. It is also important to assess the penalty for using the vectorized codes when vector lengths are short. Table 7 lists sample speedups for each function on the three machines for vectors of length 2 and 20. Arguments for these tests are distributed over two argument subranges so that internal vector lengths are actually half that listed in the table. As one would expect, the vector codes run slower in each case when very few function values are requested. For vector length 2 the vector code runs twice as long on the Sun and the Cray, and five times as long on the Convex. Table 8 lists the vector length for which the speedup is 1, i.e., the break-even point after which the vector codes are faster. The breakeven point is in the range 4–5 for the Cray, 7–11 for the Convex, and 7–14 for the Sun.

6 Conclusion

We have described a suite of portable Fortran subprograms, VFNLIB, for computing the Bessel functions of real argument and integer order. The codes are a modification of routines from Fullerton's well-known FNLIB package. The new routines allow users to specify an array of arguments at which to evaluate the functions, and the algorithms have been changed to allow effective vectorization when this is the case. Speedups of from 13 to 22

over use of the original FNLIB have been observed for the VFNLIB codes on the Cray Y-MP. Modest speedups were also observed on the Sun SPARCstation 1+, a scalar workstation. The vectorization techniques employed in VFNLIB extend to many other FNLIB routines.

Disclaimer

Identification of commercial products in this paper does not imply recommendation or endorsement by NIST.

References

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* U. S. Government Printing Office, Washington, D.C., 1964.

[2] B. L. Buzbee. The SLATEC common mathematical library. In W. R. Cowell, editor, *Sources and Development of Mathematical Software*, pages 302–318. Prentice-Hall, Englewood Cliffs, NJ, 1984.

[3] C. W. Clenshaw. A note on the summation of Chebyshev series. *M. T. A. C.*, 9:118–120, 1955.

[4] Convex Computer Corporation, Richardson, TX. *CONVEX VECLIB User's Guide*, third edition, October 1988.

[5] J. J. Dongarra. Performance of various computers using standard linear equations software. Computer Science Department Report CS–89–85, University of Tennessee, August 1990.

[6] J. J. Dongarra and E. Grosse. Distribution of mathematical software via electonic mail. *Comm. ACM*, 30:403–407, 1987.

[7] L. Fox and I. B. Parker. *Chebyshev Polynomials in Numerical Analysis.* Oxford University Press, London, 1979.

[8] P. A. Fox, A. D. Hall, and N. L. Schryer. Algorithm 528: Framework for a portable library. *ACM Trans. Math. Softw.*, 4:177–188, 1978.

[9] L. W. Fullerton. Portable special function routines. In W. Cowell, editor, *Portability of Numerical Software*, volume 57 of *Lecture Notes in Computer Science*, pages 452–483. Springer–Verlag, New York, 1976.

[10] IMSL Inc., 2500 CityWest Blvd., Houston, TX 77042-3020. *SFUN/LIBRARY: FORTRAN Subroutines for Evaluating Special Functions*, April 1987. Version 2.0.

[11] S. Wolfram. *Mathematica, A System for Doing Mathematics by Computer.* Addison-Wesley, Redwood City, CA, 1988.

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4615

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
JUNE 1991

4. TITLE AND SUBTITLE

Portable Vectorized Software for
Bessel Function Evaluation

5. AUTHOR(S)

Ronald F. Boisvert and Bonita V. Saunders

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

A suite of computer programs for the evaluation of Bessel functions and modified Bessel functions of orders zero and one for a vector of real arguments is described. Distinguishing characteristics of these programs are that (a) they are portable across a wide range of machines, and (b) they are vectorized in the case when multiple function evaluations are to be performed. The performance of the new programs are compared with software from the FNLIB collection of Fullerton on which the new software is based.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

Bessel function; hyperbolic Bessel function; mathematical software; modified Bessel function; order zero and one; portable software; special function; vectorized software

13. AVAILABILITY

[X] UNLIMITED

[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

[ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

[X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
19

15. PRICE
A02

ELECTRONIC FORM