NISTIR 6031

Parallelizing a Fourth-Order Runge-Kutta Method

Hai C. Tang

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
High Performance Systems and Services Division
Scalable Parallel Systems and Applications Group
Gaithersburg, MD 20899-0001

NIST

NISTIR 6031

Parallelizing a Fourth-Order Runge-Kutta Method

Hai C. Tang

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
High Performance Systems and Services Division
Scalable Parallel Systems and Applications Group
Gaithersburg, MD 20899-0001

June 1997

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Gary Bachula, Acting Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Robert E. Hebner, Acting Director

Parallelizing a Fourth-Order Runge-Kutta Method

Hai C. Tang
Scalable Parallel Systems and Applications Group
National Institute of Standards and Technology
Gaithersburg, Maryland

Abstract - *The most commonly used fourth-order Runge-Kutta(RK) method is examined for its suitability for parallelization. To avoid the inherent data dependence, parallelization of the RK method uses some iterations that deviate from the traditional method. Numerical results have been obtained for comparison between parallel and serial programs, and comparison with known exact solutions. The RK method is found to be parallelizable for only a small number of processors. A step control method that allows a half- or double-step sizing at run time is parallelizable for up to four processors. Serial and parallel programs are written in Fortran 90. The parallel program has a Message Passing Interface(MPI) version and a High Performance Fortran(HPF) version. The parallel methods decrease accuracy without significant gains of efficiency. When applicable, an application should be parallelized at the level calling the Runge-Kutta subroutine, with each RK invocation executed serially.*

Keywords: Runge-Kutta, parallel, scalable, MPI, HPF

1 Introduction

Runge-Kutta methods are often used to solve first-order differential equations with initial value problems:

$y' = f(x,y) \qquad x_0 \le x \le x_{end}$
$y(x_0) = y_0$
solve for $y(x_{end})$

The Runge-Kutta methods divide x_0 to x_{end} into small steps, $(x_0, x_1, ..., x_n, x_{n+1}, ..., x_{end})$ and $h_n = x_{n+1} - x_n$, and compute the value y_{n+1} at the end of each step based on the value y_n at the beginning of that step. The most commonly used fourth-order Runge-Kutta method[1-2] is as follows:

$k_1 = h_n f(x_n, y_n)$
$k_2 = h_n f(x_n + \frac{1}{2}h_n, y_n + \frac{1}{2}k_1)$
$k_3 = h_n f(x_n + \frac{1}{2}h_n, y_n + \frac{1}{2}k_2)$
$k_4 = h_n f(x_n + h_n, y_n + k_3)$
$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

These formulas indicate that the fourth-order Runge-Kutta method computes the value $h_n * f(x,y)$ for $x_n \le x \le x_{n+1}$ at four selected (x,y) pairs and averages them with weighted coefficients.

This method explicitly computes the k_i and y_n values, but data dependence also exists between formulas. In this case, k_4 depends on k_3, k_3 on k_2 and k_2 on k_1, and y_{n+1} depends on y_n and all k values. These data dependencies hinder the parallel operation of the method, though there have been efforts to parallelize the Runge-Kutta methods[3-4].

2 Parallelizing Runge-Kutta Method

To parallelize the fourth-order Runge-Kutta method, we consider the following cases:

- If $f(x,y) = f(x)$, the above fourth-order Runge-Kutta method becomes a third order Runge-Kutta method, also known as the Simpson rule of integration. Furthermore, there exists no data dependence. The computation of k_i for i=1,..,4 for all steps can be parallelized, and the value $y(x_{end})$ can also be computed using the intrinsic SUM function. This allows parallel computing for any number of processors.

- If $f(x,y) \approx f(x,y_n)$ in the interval $x_n \leq x \leq x_{n+1}$, the fourth-order Runge-Kutta method can be modified as follows:

  $g_1 = f(x_n, y_n)$
  $g_2 = f(x_n + \frac{1}{2}h_n, y_n + \frac{1}{2}g_2 * h_n)$
  $g_3 = f(x_n + \frac{1}{2}h_n, y_n + \frac{1}{2}g_3 * h_n)$
  $g_4 = f(x_n + h_n, y_n + g_4 * h_n)$
  $y_{n+1} = y_n + \frac{1}{6}(g_1 + 2g_2 + 2g_3 + g_4) * h_n$

  Similiar to iterative methods in [3-4], the g_i for i=2,4 are initially set to g_1. The first iteration of g_2 is exactly k_2/h_n, and the second iteration of g_2 is exactly k_3/h_n. Value g_4 can be iteratively computed a certain number of times to obtain a converged value. It will be seen that g_4 converges in four iterations.

  Since k_2 and k_3 are the values of the first two iterations of g_2 times h_n, respectively, the above single step fourth-order Runge-Kutta parallel method can be described as the following:

  $g_1 = f(x_n, y_n)$
  $g_2 = f(x_n + \frac{1}{2}h_n, y_n + \frac{1}{2}g_2 * h_n)$
  $g_4 = f(x_n + h_n, y_n + g_4 * h_n)$
  $y_{n+1} = y_n + \frac{1}{6}(g_1 + 2(g_2^1 + g_2^2) + g_4) * h_n$

  The values of g_2 and g_4 are initially set to g_1, then g_2 is iterated twice (g_2^1 and g_2^2) and g_4 is iterated four times. Thus, only the computation of g_2 and g_4 can be parallelized. This requires only two parallel processors. The parallel program, that computes g_1 once and then iteratively computes g_4 four times in one processor, does more computation than the serial program that computes once each of k_1,k_2, k_3, and k_4 in a single processor. In addition, the parallel program requires synchronization and time to transmit either g_2 or g_4 to the processor to compute y_{n+1}. Timing results show that the HPF parallel program running on IBM SP2 for two processors is actually about 33% slower than the serial program on a single processor.

The parallel method, described above, is different from the traditional serial fourth-order RK method in that its g_4 is iterated based on the values (x_{n+1}, y_n), instead of being computed from g_2^2, to avoid data dependence. It will be seen that the value g_4 converges in four iterations as stated in [3], but the value does not converge to $k4/h_n$. We have not analyzed its mathematical order but, based on the examples in Section 4, it appears to be less accurate than the traditional fourth-order RK method.

- The assumption that $f(x,y) \approx f(x,y_n)$ can also be extended to multiple steps or intervals. Initially $f(x,y)$ values for the whole block from x_n to x_{n+l} are set to $f(x,y_n)$ and are computed in parallel using an iterative method. Thus a higher speedup may be obtained. However, the accuracy of results from this multi-step method decreases rapidly as the number of steps(l value) in the block increases.

3 Runge-Kutta Step Control Method

Assuming that the parallel Runge-Kutta variants remain fourth-order, the interval between steps may be changed at run time. A step control method that allows half- or double-step sizing at run time[2] can be parallelized for up to four processors. This method may be derived from the single step method.

The fourth-order Runge-Kutta method has an asymptotic error of the form[2]:

$$y_1(x_{n+1}) = y(x_{n+1}) + C(x_{n+1})h^4 + O(h^5)$$

The fourth-order Runge-Kutta method can now be integrated from x_n to x_{n+1} ($x_{n+1} = x_n + h$) twice, once using a step of length h and again using two steps of length $h/2$. In this paper, these two computed $y(x_{n+1})$ values are denoted as $y_1(x_{n+1})$ and $y_2(x_{n+1})$, respectively.

$$y_2(x_{n+1}) = y(x_{n+1}) + C(x_{n+1})(\tfrac{h}{2})^4 + O((\tfrac{h}{2})^5)$$

By ignoring the highest order, one can obtain the corrector formula:

$$y(x_{n+1}) = y_2(x_{n+1}) + \tfrac{1}{2^4-1}(y_2(x_{n+1}) - y_1(x_{n+1})).$$

The following formula can therefore be used to calculate the estimated error:

$$D_{n+1} = \tfrac{1}{2^4-1}|y_2(x_{n+1}) - y_1(x_{n+1})|$$

One can then use D_{n+1} to decide whether the current step is within the desired limits. If D_{n+1} is too large, the step size is divided in half. If D_{n+1} is consistently too small, the next step size is doubled. To decide whether D is consistently small in order to avoid a half stepping immediately followed by a double stepping, D is defined as:

$$D = D_{n+1} + 0.1 * D_n + 0.01 * D_{n-1}$$

The size of the next step can then be decided as follows:

$\epsilon' \leq D \leq \epsilon$	Continue with the same step size
$D > \epsilon$	Use the $y_2(x_{n+h/2})$ value and continue with the half step size
$D < \epsilon'$	Continue next step with the double step size

For a single precision floating point computation, ϵ is set to 1.0E-6, and ϵ' to 0.02*ϵ.

A parallel method can be derived from the single step method. Each step is equally divided into four substeps. These are the corresponding g_i functions:

$g_1 = f(x_n, y_n)$
$g_2 = f(x_n + \frac{1}{4}h_n, y_n + \frac{1}{4}g_2 * h_n)$
$g_3 = f(x_n + \frac{1}{2}h_n, y_n + \frac{1}{2}g_3 * h_n)$
$g_4 = f(x_n + \frac{3}{4}h_n, y_n + \frac{3}{4}g_4 * h_n)$
$g_5 = f(x_n + h_n, y_n + g_5 * h_n)$

The values of g_i for i=2 to 5 are initially set to g_1. Then for i=2 to 3, g_i are iterated twice and denoted as g_i^1, g_i^2, respectively. Both g_4 and g_5 are iterated four times to obtain converged values before they are used to compute $y_1(x_{n+1})$ and $y_2(x_{n+1})$.

$y_1(x_{n+1}) = y_n + \frac{1}{6}(g_1 + 2(g_3^1 + g_3^2) + g_5) * h_n$

$y_2(x_{n+1}) = y_n + \frac{1}{12}(g_1 + 2(g_2^1 + g_2^2) + (g_3^1 + g_3^2) + 4g_4 + g_5) * h_n$

$D_{n+1} = \frac{1}{180}|(2(g_2^1 + g_2^2) + 4g_4 - g_1 - 3(g_3^1 + g_3^2) - 3(g_3^1 + g_3^2) - g_5) * h_n|$

For i=2 to 5, the computation of g_i can be parallelized for up to four processors. As g_4 and g_5 are iterated based on (x_n, y_n) instead of $(x_{n+1/2}, y_{n+1/2})$, the corrector formula is not used to revise $y(x_{n+1})$ value.

Since the parallel method is less accurate than the serial method, ϵ is set to 5.0E-5 and ϵ' to $0.02^*\epsilon$.

4 Comparison of Results

Serial and parallel programs are written in Fortran 90, and the parallel program is implemented with both High Performance Fortran(HPF) and Message Passing Interface(MPI). The HPF version distributes the computation of g_i to separate processors by the distribute(block) directive. The MPI version does the same computation with data communication via mpi_bcast, mpi_barrier, and mpi_gather. A scalable program that parallelizes at the level calling the Runge-Kutta subroutine and serially executes the subroutine in separate processors is implemented with MPI group communication routines and point-to-point communication routines. The source codes are listed in Appendices.

Consider the following first order differential equation:
$y' = f(x, y) = 1/x * *2 - y/x - y * *2$
with an initial value $y(1.0) = 1.0$,
solve $y(x)$ for $1.0 \leq x \leq 3.0$.

This initial value problem has an exact solution of:
$y(x) = 1/x$

The range form $x = 1$ to $x = 3$ is divided into 40 equal steps, and the input data for the serial program are:

N	x_0	x_{end}	$y(x_0)$
40	1.0	3.0	1.0

Table 1 shows some selected results of the serial program. It gives the same solution as the exact solution at up to six effective digits.

Table 1. Serial Program Results.

XN	YN	Y=1/X	YN-Y
1.00	1.000000	1.000000	0.0000E+0
1.20	0.833333	0.833333	0.1192E-6
1.40	0.714286	0.714286	0.1192E-6
1.60	0.625000	0.625000	0.1788E-6
1.80	0.555556	0.555556	0.1192E-6
2.00	0.500000	0.500000	0.1192E-6
2.20	0.454546	0.454545	0.8941E-7
2.40	0.416667	0.416667	0.5960E-7
2.60	0.384615	0.384615	0.2980E-7
2.80	0.357143	0.357143	0.5960E-7
3.00	0.333333	0.333333	0.2980E-7

For the uniformly parallel program, both the number of steps in the block and the number of iterations, NS and PM respectively, are varied. The following inputs are used.

N	NS	PM	x_0	x_{end}	$y(x_0)$
40	1	2	1.0	3.0	1.0
40	1	3	1.0	3.0	1.0
40	1	4	1.0	3.0	1.0
40	1	6	1.0	3.0	1.0
40	2	2	1.0	3.0	1.0
40	2	3	1.0	3.0	1.0
40	2	4	1.0	3.0	1.0
40	2	8	1.0	3.0	1.0
40	4	4	1.0	3.0	1.0

As the single step parallel method deviates from the traditional method in the computation of g_4, it is interested to compare the difference between g_4 and $k4/h$. The following table shows iteration of $g_4(PM)$ converge at $PM = 4$, but it does not converge to $k4/h$. They differ starting from the third effective digit.

Table 2. The convergence of g_4 iterations and its comparison with $k4/h$.

XN	YN	G4(1)	G4(2)	G4(3)	G4(4)	G4(5)	K4/H
1.00	1.00000	-.90023	-.91449	-.91245	-.91274	-.91270	-.90684
1.25	0.80000	-.58888	-.59478	-.59410	-.59417	-.59417	-.59166
1.50	0.66667	-.41485	-.41771	-.41743	-.41746	-.41746	-.41621
1.75	0.57143	-.30789	-.30944	-.30931	-.30932	-.30932	-.30863
2.00	0.50000	-.23751	-.23842	-.23836	-.23836	-.23836	-.23795
2.25	0.44445	-.18876	-.18933	-.18929	-.18929	-.18929	-.18903
2.50	0.40000	-.15360	-.15398	-.15396	-.15396	-.15396	-.15379
2.75	0.36364	-.12743	-.12768	-.12767	-.12767	-.12767	-.12755

The iterations of g_i $(i > 1)$ for multi-steps also show that all values of g_i converge at $PM = 4$.

Table 3 shows the results of the parallel program for three different multi-steps. For a single step(NS=1) and two steps(NS=2), the results have a percentage error at least two orders of magnitude greater than that induced by the serial method. For four steps(NS=4), the errors are less than 1%, which may be allowable in some applications. Thus the fourth-order Runge-Kutta method may be parallelizable for up to eight processors in some applications, but the number of parallel processors may preferably be limited to four.

Table 3. Parallel Program Results.

XN	YN(NS=1)	YN(NS=2)	YN(NS=4)
1.00	1.000000	1.000000	1.000000
1.20	0.833210	0.832401	0.828857
1.40	0.714142	0.713204	0.709280
1.60	0.624865	0.623989	0.620424
1.80	0.555437	0.554667	0.551596
2.00	0.499898	0.499236	0.496631
2.20	0.454458	0.453892	0.451686
2.40	0.416592	0.416107	0.414234
2.60	0.384551	0.384134	0.382534
2.80	0.357087	0.356727	0.355352
3.00	0.333285	0.332973	0.331783

average/maximum |% error|		
0.018/0.022	0.14/0.16	0.61/0.73

For a single precision floating point computation, the serial program could have results that are accurate up to the sixth effective digits. The parallel method, however, could be accurate only up to four digits for NS=1 and NS=2. The accuracy of results degrades rapidly and becomes unacceptable as NS increases beyond 4.

The input for step control method would be:

h_0	x_0	x_{end}	$y(x_0)$
0.2	1.0	3.0	1.0

The step control method for the serial program gives the solution accuracy up to six effective digits of the exact solution(Table 4). The parallel program has an error of less than 0.1% (Table 5). The wall clock execution time of the four-processor parallel program is only 25% less than that of the single processor serial program. The parallel programs decrease accuracy without significant gains of efficiency.

Table 4. Test Results from the Serial Program with Step Control

N	XN	YN	Y=1/X	D	H
0	1.000000	1.000000	1.000000	0.0000E+00	0.2000
1	1.100000	0.909093	0.909091	0.3437E-05	0.1000
2	1.200000	0.833335	0.833333	0.8345E-07	0.1000
3	1.300000	0.769232	0.769231	0.4768E-07	0.1000
4	1.400000	0.714287	0.714286	0.3179E-07	0.1000
5	1.500000	0.666668	0.666667	0.1987E-07	0.1000
6	1.600000	0.625001	0.625000	0.1192E-07	0.2000
7	1.800000	0.555556	0.555556	0.2464E-06	0.2000
8	2.000000	0.500000	0.500000	0.1272E-06	0.2000
9	2.200000	0.454546	0.454545	0.6954E-07	0.2000
10	2.400000	0.416667	0.416667	0.4172E-07	0.2000
11	2.600000	0.384616	0.384615	0.2384E-07	0.2000
12	2.800000	0.357143	0.357143	0.1589E-07	0.4000
13	3.000000	0.333333	0.333333	0.9934E-08	0.4000

Table 5. Test Results from the Parallel Program with Step Control

N	XN	YN	Y=1/X	D	H
0	1.000000	1.000000	1.000000	0.0000E+00	0.2000
1	1.100000	0.909030	0.909091	0.7983E-04	0.1000
2	1.200000	0.833054	0.833333	0.1001E-04	0.1000
3	1.300000	0.768843	0.769231	0.7260E-05	0.1000
4	1.400000	0.713851	0.714286	0.5396E-05	0.1000
5	1.500000	0.666220	0.666667	0.4097E-05	0.1000
6	1.600000	0.624559	0.625000	0.3167E-05	0.1000
7	1.700000	0.587811	0.588235	0.2488E-05	0.1000
8	1.800000	0.555153	0.555556	0.1975E-05	0.1000
9	1.900000	0.525937	0.526316	0.1593E-05	0.1000
10	2.000000	0.499646	0.500000	0.1297E-05	0.1000
11	2.100000	0.475860	0.476190	0.1067E-05	0.1000
12	2.200000	0.454238	0.454545	0.8861E-06	0.1000
13	2.300000	0.434496	0.434783	0.7411E-06	0.2000
14	2.500000	0.399679	0.400000	0.4236E-05	0.2000
15	2.700000	0.370044	0.370370	0.3113E-05	0.2000
16	2.900000	0.344510	0.344828	0.2340E-05	0.2000
17	3.000000	0.333041	0.333333	0.2543E-06	0.2000

mean/max. |% error|
0.066/0.092

Table 6 shows the average and maximum absolute percentage errors of the parallel programs for three differential equations from $x_0 = 1.0$, to $x_{end} = 4.0$. It is worth mentioning that the errors for four-processor step control method are within 0.1%.

Table 6. Average/Maximum absolute % of error by the Parallel Program for 3 differential equations.

Diff. Equ./ Exact Solu.	Uniform Steps $NS = 1$	2	4	Control Steps (h_0=0.05)	
$y' = x + y$	0.044	0.33	1.50	0.044	*avg*
$y = e^x - x - 1$	0.071	0.53	2.34	0.071	*max*
$y' = x - y$	0.002	0.012	0.05	0.015	*avg*
$y = e^{-x} + x - 1$	0.003	0.024	0.10	0.029	*max*
$y' = 1 - y/x$	0.006	0.043	0.18	0.015	*avg*
$y = x/2 + 2/x$	0.008	0.057	0.24	0.024	*max*

The parallel methods decrease accuracy without significant gains of efficiency. When applicable, an application should be parallelized at the level calling the Runge-Kutta subroutine and serially execute the subroutine in sparate processors. For example, parallelize the following problem in spatial dimensions, but serially compute the Runge-Kutta integration in time sequence in each processor[5]. In this way, the execution may scale to the complexity of the problem.

$$y'(x,t) = f(x,t,y(x,t)) \qquad x_0 \le x \le x_{end}$$
$$t_0 \le t \le t_{end}$$
$$y(x,t_0) = y_0(x)$$

Such a scalable program is implemented with MPI group communication routines and point-to-point communication routines.

5 Summary and Conclusions

The most commonly used fourth-order Runge-Kutta method has been closely studied for parallel computation. Three parallel methods are developed and implemented, but their results are less accurate than what the serial programs produces. While a uniformly spaced single step Runge-Kutta method can be parallelized only for two processors, a uniformly-spaced multi-step parallel Runge-Kutta method can be parallelized for up to eight processors. However, the error increases rapidly with the increasing number of processors. The eight-processor parallel program could result in a 1% error. A step control method based on the fourth-order Runge-Kutta method allows half- or double-step sizing at run time and can be parallelized for up to four processors with a possible error up to 0.1%. Both serial and parallel versions of the fourth-order Runge-Kutta method are written in Fortran 90, and the parallel program has both HPF and MPI versions.

The parallel methods decrease accuracy without significant gains of efficiency. The four-processor parallel program is only 25% faster than the single processor serial program. The error induced by either parallel method is at least two orders of magnitude greater than that induced by the serial method. For a single precision floating point computation, the serial

program could have a precision of up to six effective digits. The parallel method, however, could have a precision of at best up to four digits.

The Runge-Kutta method itself requires little memory and uses little CPU time. Most CPU time is most likely spent on the computation of $f(x, y)$ which is supplied by the user. It is questionable whether the parallel program has an advantage over the serial program. In a scalable application of the Runge-Kutta method, instead of parallelizing the computation inside the subroutine, one should parallelize the work above the subroutine and have the subroutine executed serially in separate processors. In this way, the execution may scale to the complexity of the problem.

References

[1] Anthony Ralston. A First Course in Numerical Analysis. McGraw-Hill Book Company, News York, 1965.

[2] S. D Conte and Carl de Boor. Elementary Numerical Analysis, an Algorithmic Approach. McGraw-Hill Book Company, New York, 1980.

[3] K. R. Jackson, and S. P. Norsett. "The Potential for Parallelism in Runge-Kutta Methods. Part I: RK Formulas in Standard Form." SIAM Journal of Numerical Analysis, Feb 1995.

[4] Thomas Rauber, and Gudula Runger. "Iterated Runge-Kutta Methods on Distributed Memory Multiprocessors." Proc. of 3rd Euromicro Workshop on Parallel and Distributed Processing, 1995.

[5] Gilbert Strang. Introduction to Applied Mathematics. Wellesley-Cambridge Press, Massachusetts, 1986.

[6] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele Jr., and M. E. Zosel. The High Performance Fortran Handbook. The MIT Press, Cambridge, Massachusetts, 1994.

[7] W. Gropp, E. Lusk, and A. Skjellum. Using MPI: Portable Parallel Programming with the Message-Passing Interface. The MIT Press, Cambridge, Mass., 1996.

[8] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. MPI: The Complete Reference. The MIT Press, Cambridge, Mass., 1996.

Appendix A. Serial Implementation

```
! Uniform step size
! Fourth-Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                              Y(XBEGIB) = YBEGIN
! Solve for Y at the end point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      INTEGER NSTEPS
      REAL XBEGIN,XEND,YBEGIN,YEND
      REAL, EXTERNAL :: FXY
      READ(*,*) NSTEPS,XBEGIN,XEND,YBEGIN
  501 FORMAT(I6,3F10.4)
      YEND = RK4S(FXY,NSTEPS,XBEGIN,XEND,YBEGIN)
      WRITE(*,'(A,F10.6,A,E14.6)') "Y(",XEND," ) =",YEND
      STOP
      END
      FUNCTION RK4S(F,NSTEPS,XBEGIN,XEND,YBEGIN)
      INTEGER N,NSTEPS
      REAL DERIV,H,K1,K2,K3,K4,XBEGIN,XN,XEND,YBEGIN,YN
      IF(NSTEPS.LT.1) NSTEPS=1
      H = (XEND-XBEGIN)/NSTEPS
      XN = XBEGIN
      YN = YBEGIN
      DERIV = F(XN,YN)
      R6 = 1.0/6
      N=0
!     WRITE(*,"(4X,A,A)") " N         XN           YN        YX",
!    &                    "       YN-YX       DERIV "
!     WRITE(*,601) N,XN,YN,YN,YN-YN,DERIV
! 601 FORMAT(I6,3F12.6,3E12.4)
      DO 10 N=1,NSTEPS
         K1 = H*DERIV
         K2 = H*F(XN+0.5*H,YN+0.5*K1)
         K3 = H*F(XN+0.5*H,YN+0.5*K2)
         K4 = H*F(XN+H,YN+K3)
         YN = YN +R6*(K1+2*K2+2*K3+K4)
         XN = XBEGIN + N*H
         YN0 = YBEGIN/XN
         DERIV = F(XN,YN)
!        WRITE(*,601) N,XN,YN,YN0,YN-YN0,DERIV
   10    CONTINUE
      RK4S=YN
      RETURN
      END
      REAL FUNCTION FXY(X,Y)
      REAL X,Y
      FXY= 1.0/X**2 - Y/X -Y*Y
      RETURN
      END
```

```
! Runge-Kutta Step Control Method
! Runge-Kutta Predictor-Corrector Method
! with Half or Double step sizing at run time.
! Fourth-Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                            Y(XBEGIB) = YBEGIN
! Solve for Y at the end point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      REAL STEP,XBEGIN,XEND,YBEGIN,YEND
      REAL, EXTERNAL :: FXY
      READ(*,*) STEP,XBEGIN,XEND,YBEGIN
  501 FORMAT(6F10.4)
      YEND = RK4SC(FXY,STEP,XBEGIN,XEND,YBEGIN)
      WRITE(*,'(A,F10.6,A,E14.6)') "Y(",XEND," ) =",YEND
      STOP
      END
      FUNCTION RK4SC(F,STEP,XBEGIN,XEND,YBEGIN)
      INTEGER N
      REAL DERIV,STEP,H,K1,K2,K3,K4,XBEGIN,XN,XEND,YBEGIN,YN
      REAL EPS,R6,D,DD,HH,YH,YHH
      XN = XBEGIN
      YN = YBEGIN
      DERIV = F(XN,YN)
      EPS=1.0E-6
      R6 = 1.0/6
      R15 = 1.0/15.0
      H = STEP
      DD = 0.1*H*EPS
      N=0
!     WRITE(*,"(4X,A,A)") " N         XN          YN1         YN2",
!    &                    "       DERIV           D           H"
!     WRITE(*,601) N,XN,YN,YN,DERIV
  601 FORMAT(I6,3F12.6,3E12.4)
      DO WHILE(XN.LE.XEND)
         IF(ABS(XN-XEND) .LT. 1.0E-6) EXIT
         XNH = XN + H
         IF(XNH.GT.XEND) THEN
           H=XEND-XN
           XNH=XEND
         ENDIF
         N = N+1
         K1 = H*DERIV
         K2 = H*F(XN+0.5*H,YN+0.5*K1)
         K3 = H*F(XN+0.5*H,YN+0.5*K2)
         K4 = H*F(XN+H,YN+K3)
         YH = YN +R6*(K1+2*K2+2*K3+K4)
    !!   HALF STEP-SIZE
         HH = 0.5*H
         K1 = 0.5*K1
         K2 = HH*F(XN+0.5*HH,YN+0.5*K1)
         K3 = HH*F(XN+0.5*HH,YN+0.5*K2)
```

```
      K4 = HH*F(XN+HH,YN+K3)
      YH1 = YN +R6*(K1+2*K2+2*K3+K4)
      K1 = HH*F(XN+HH,YH1)
      K2 = HH*F(XN+1.5*HH,YH1+0.5*K1)
      K3 = HH*F(XN+1.5*HH,YH1+0.5*K2)
      K4 = HH*F(XN+2.0*HH,YH1+K3)
!!      Predictor and corrector
      YHH = YH1 +R6*(K1+2*K2+2*K3+K4)
      D = R15*(YHH-YH)
      YNH = YHH + D
      DD = 0.1*DD + ABS(D/H)
      IF(DD.GT.H*EPS) THEN
!!      HALF STEP SIZING
        H=HH
        XNH = XN + H
        YNH=YH1
      ELSE
!!      DOUBLE STEP SIZING
        IF(DD.LT.H*EPS*0.02) H=2*H
      ENDIF
!     WRITE(*,601) N,XNH,YH,YNH,DERIV,D,H
      XN = XNH
      YN = YNH
      DERIV = F(XN,YN)
      END DO
    RK4S=YN
    RETURN
    END
    REAL FUNCTION FXY(X,Y)
    REAL X,Y
    FXY = 1.0/X**2 - Y/X -Y*Y
    RETURN
    END
```

Appendix B. Parallel Implementation

```
! Uniform step size
! Fourth-Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                              Y(XBEGIB) = YBEGIN
! Solve for Y at the point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      INTEGER NSTEPS,PSTEPS
      REAL XBEGIN,XEND,YBEGIN,YEND
      REAL, EXTERNAL :: FXY
      READ(*,*) NSTEPS,PSTEPS,XBEGIN,XEND,YBEGIN
! 501 FORMAT(2I6,3F10.4)
      YEND = RK4S(FXY,NSTEPS,PSTEPS,XBEGIN,XEND,YBEGIN)
      WRITE(*,'(A,F10.6,A,E14.6)') "Y(",XEND," ) =",YEND
      STOP
      END
      FUNCTION RK4S(F,NSTEPS,PSTEPS,XBEGIN,XEND,YBEGIN)
      INTEGER N,NSTEPS,PSTEPS,PS2,PM,I
      REAL XT(PSTEPS*2),G(PSTEPS*2)
      REAL H,HH,XN,YN,GN,XBEGIN,XEND,YBEGIN
!HPF$ DISTRIBUTE (BLOCK) :: XT,G
      IF(NSTEPS.LT.1) NSTEPS=1
      IF(PSTEPS.LT.1) PSTEPS=1
      PS2 = PSTEPS*2
      PM = 3
      H = (XEND-XBEGIN)/NSTEPS
      HH = 0.5*H
      XN = XBEGIN
      YN = YBEGIN
      GN = F(XN,YN)
      R6 = 1.0/6
      N=0
      WRITE(*,"(4X,A,A)") " N         XN           YN        YX",
     &                    "       YN-YX       DERIV "
      WRITE(*,601) N,XN,YN,YN,YN-YN,G(1)
  601 FORMAT(I6,3F12.6,3E12.4)
      DO 100 N=1,NSTEPS,PSTEPS
      DO 10 I=1,PS2
         G(I) = GN
         XT(I)= XN+HH*I
   10    CONTINUE
      DO 20 I=1,PS2
      DO 21 M=1,PM
         G(I) = F(XT(I),YN+G(I)*HH*I)
   21    CONTINUE
   20    CONTINUE
      DO 30 I=2,PS2,2
         G(I) = 4.0*G(I)
   30    CONTINUE
      DO 32 I=3,PS2,2
         G(I) = 2.0*G(I)
```

```
   32    CONTINUE
         SUM = G1
         DO 40 I=1,PS2
            SUM = SUM + G(I)
   40    CONTINUE
         YN = YN + R6*SUM*H
         XN = XN + PSTEPS*H
         YN0 = YBEGIN/XN
         GN = F(XN,YN)
         WRITE(*,601) N,XN,YN,YN0,YN-YN0,GN
  100 CONTINUE
         RK4S=YN
         RETURN
         END
         REAL FUNCTION FXY(X,Y)
         REAL X,Y
         FXY= 1.0/X**2 - Y/X -Y*Y
         RETURN
         END
```

```
! Runge-Kutta Step Control Method
! Runge-Kutta Predictor-Corrector Method
! with Half or Double step sizing at run time.
! Fourth-Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                            Y(XBEGIB) = YBEGIN
! Solve for Y at the point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      REAL STEP,XBEGIN,XEND,YBEGIN,YEND
      REAL, EXTERNAL :: FXY
      READ(*,*) STEP,XBEGIN,XEND,YBEGIN
  501 FORMAT(6F10.4)
      YEND = RK4SC(FXY,STEP,XBEGIN,XEND,YBEGIN)
      WRITE(*,'(A,F10.6,A,E14.6)') "Y(",XEND," ) =",YEND
      STOP
      END
      FUNCTION RK4SC(F,STEP,XBEGIN,XEND,YBEGIN)
      INTEGER N,I,M,PM
      REAL XT(4),G(4)
      REAL GN,STEP,H,HH,HQ,XBEGIN,XEND,YBEGIN,XN,YN
      REAL EPS,R6,D,DD,XNH,YNH,YH1,YH2
!HPF$ DISTRIBUTE (BLOCK) :: XT,G
      EPS=5.0E-5
      R6 = 1.0/6
      R15 = 1.0/15.0
      R180 = 1.0/180.0
      PM = 3
      XN = XBEGIN
      YN = YBEGIN
```

```
    GN = F(XN,YN)
    H = STEP
    HH=0.5*H
    D = 0.0
    N=0
    WRITE(*,"(4X,A,A)") " N       XN           YN           Y_2",
   &                     "          y_1          D            H"
    WRITE(*,601) N,XN,YN,YN,YN,D,H
601 FORMAT(I6,4F12.6,2E12.4)
    DO WHILE(XN.LE.XEND)
       IF(ABS(XN-XEND) .LT. 1.0E-6) EXIT
       IF(H.LT.0.01*STEP) THEN
         WRITE(*,"(A,E12.6,A)") " H =",H," < 0.01*Initial_Step_Size."
         EXIT
       ENDIF
       XNH = XN + H
       IF(XNH.GT.XEND) THEN
         H=XEND-XN
         XNH=XEND
       ENDIF
       HQ = 0.25*H
       DD = 0.1*H*EPS
       N = N+1
       DO 10 I=1,4
          G(I) = GN
          XT(I)= XN+HQ*I
 10    CONTINUE
       DO 20 I=1,2
       G(I) = F(XT(I),YN+G(I)*HQ*I)
       G(I) = G(I) + F(XT(I),YN+G(I)*HQ*I)
 20    CONTINUE
       DO 22 I=3,4
       DO 21 M=1,PM
          G(I) = F(XT(I),YN+G(I)*HQ*I)
 21    CONTINUE
 22    CONTINUE
       WRITE(*,'(A,5E12.6)') "GN,2G(1),2G(2),G(3),G(4)=",GN,
   &                          (G(I),I=1,4)
       YH1 = YN +R6*(GN+2*G(2)+G(4))*H
       HH = 0.5*H
       YH2 = YN +R6*(GN+2.0*G(1)+G(2)+4.0*G(3)+G(4))*HH
  !!     Predictor and corrector
  !!   D = R15*(YH2-YH1)
       D = R180*(2.0*G(1)+4.0*G(3)-GN-3.0*G(2)-G(4))*H
       YNH = YH2 + D
       DD = 0.1*DD + ABS(D)
       IF(DD.GT.H*EPS) THEN
  !!     HALF STEP SIZING
         H=HH
         XNH = XN + H
         YNH = YN +R6*(GN+2.0*G(1)+0.5*G(2))*HH
       ELSE
  !!     DOUBLE STEP SIZING
         IF(DD.LT.H*EPS*0.02) H=2*H
```

```
         ENDIF
         YHX = YBEGIN/XNH
         WRITE(*,601) N,XNH,YNH,YH2,YH1,D,H
         XN = XNH
         YN = YNH
         GN = F(XN,YN)
         END DO
       RK4S=YN
       RETURN
       END
       REAL FUNCTION FXY(X,Y)
       REAL X,Y
       FXY = 1.0/X**2 - Y/X -Y*Y
       RETURN
       END
```

```
! Uniform step size
! Fourth Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                            Y(XBEGIB) = YBEGIN
! to the point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      INTEGER NSTEPS,PSTEPS,PM
      REAL XBEGIN,XEND,YBEGIN,YEND
      REAL, EXTERNAL :: FXY
      OPEN(10,FILE="rk4data2")
      READ(10,501) NSTEPS,PSTEPS,PM,XBEGIN,XEND,YBEGIN
  501 FORMAT(3I6,3F10.4)
      YEND = RK4S(FXY,NSTEPS,PSTEPS,PM,XBEGIN,XEND,YBEGIN)
      WRITE(*,'(A,F10.6,A,E14.6)') "Y(",XEND," ) =",YEND
      STOP
      END
      FUNCTION RK4S(F,NSTEPS,PSTEPS,PM,XBEGIN,XEND,YBEGIN)
      include "mpif.h"
      INTEGER N,NSTEPS,PSTEPS,PS2,PM,I,size,rank,ierr
      REAL XT,G,G0(PSTEPS*2),DBLK(4)
      REAL H,HH,XN,YN,GN,XBEGIN,XEND,YBEGIN
      CALL MPI_INIT(ierr)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
      WRITE(*,"(A,I4,A,I4)") "Size=",size,"   rank= ",rank
      IF(NSTEPS.LT.1) NSTEPS=1
      IF(PSTEPS.LT.1) PSTEPS=1
      IF(MOD(size,2).NE.0) THEN
        WRITE(*,'(A)') "Number of Processors must be EVEN",              &
     &                 "Execution aborted!!"
        STOP
      ENDIF
```

```
      PS2 = PSTEPS*2
      IF(PS2.GT.size) THEN
        PS2 = size
        PSTEPS = PS2/2
      ENDIF
!!    PM = 4
      H = (XEND-XBEGIN)/NSTEPS
      HH = 0.5*H
      XN = XBEGIN
      YN = YBEGIN
      GN = F(XN,YN)
      R6 = 1.0/6
      N=0
      SPERR = 0.0
      WRITE(*,"(4X,A,A)") " N         XN           YN         YX"           &
     &                      ,"    100(YN-YX)/YX    "
      WRITE(*,601) N,XN,YN,YN,YN-YN
  601 FORMAT(I6,4F12.6,3E12.4)
      DO 100 N=1,NSTEPS,PSTEPS
         I = rank + 1
         G = GN
         XT= XN+HH*I
         IF(I.EQ.1) THEN
           G = F(XT,YN+G*HH)
           G = G + F(XT,YN+G*HH)
         ELSE
           DO 21 M=1,PM
              G = F(XT,YN+G*HH*I)
   21      CONTINUE
           IF(MOD(I,2).EQ.1) G = G*2.0
         ENDIF
         IF(I.LT.size) G = G*2.0
         CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
         CALL MPI_GATHER(G,1,MPI_REAL,G0,1,MPI_REAL,0,MPI_COMM_WORLD,    &
     &       ierr)
         IF(rank.EQ.0) THEN
           SUM = GN
           DO 40 I=1,PS2
              SUM = SUM + G0(I)
   40      CONTINUE
           YN = YN + R6*SUM*H
           XN = XN + PSTEPS*H
  !        YN0 = XN/2 + 2/XN
           YN0 = YBEGIN/XN
           PERR = 100.0*(YN-YN0)/YN0
           SPERR = SPERR + ABS(PERR)
           GN = F(XN,YN)
           WRITE(*,601) N,XN,YN,YN0,PERR
           DBLK(1) = XN
           DBLK(2) = YN
           DBLK(3) = GN
           CALL MPI_BCAST(DBLK,3,MPI_REAL,0,MPI_COMM_WORLD,ierr)
         ELSE
           CALL MPI_BCAST(DBLK,3,MPI_REAL,0,MPI_COMM_WORLD,ierr)
```

```
          XN = DBLK(1)
          YN = DBLK(2)
          GN = DBLK(3)
        ENDIF
  100 CONTINUE
      WRITE(*,'(A,F10.4)') "ACC ERROR =",SPERR
      CALL MPI_FINALIZE(ierr)
      RK4S=YN
      RETURN
      END
      REAL FUNCTION FXY(X,Y)
      REAL X,Y
  !   FXY = 1.0 - Y/X
      FXY= 1.0/X**2 - Y/X -Y*Y
      RETURN
      END



! Runge-Kutta Step Control Method
! Runge-Kutta Predictor-Corrector Method
! with Half or Double step sizing at run time.
! Fourth Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                            Y(XBEGIB) = YBEGIN
! to the point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      REAL STEP,XBEGIN,XEND,YBEGIN,YEND
      REAL, EXTERNAL :: FXY
      OPEN(10,FILE="rk4data")
      READ(10,501) STEP,XBEGIN,XEND,YBEGIN
  501 FORMAT(6F10.4)
      YEND = RK4SC(FXY,STEP,XBEGIN,XEND,YBEGIN)
      WRITE(*,'(A,F10.6,A,E14.6)') "Y(",XEND," ) =",YEND
      STOP
      END
      FUNCTION RK4SC(F,STEP,XBEGIN,XEND,YBEGIN)
      include "mpif.h"
      INTEGER N,I,M,PM,size,rank,ierr
      REAL XT,G,G0(4)
      REAL GN,STEP,H,HH,HQ,XBEGIN,XEND,YBEGIN,XN,YN
      REAL EPS,R6,D,DD,XNH,YNH,YNX,YH2
      CALL MPI_INIT(ierr)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
      WRITE(*,"(A,I4,A,I4)") "Size=",size,"   rank= ",rank
      EPS = 5.0E-5
      R6 = 1.0/6
      R15 = 1.0/15.0
      PM = 4
```

```
    XN = XBEGIN
    YN = YBEGIN
    GN = F(XN,YN)
    H = STEP
    HH=0.5*H
    HQ = 0.25*H
    DD = 0.1*H*EPS
    D = 0.0
    N=0
    WRITE(*,"(4X,A,A)") " N       XN           YN           YNX",          &
   &                     "          y_2          D            H"
    WRITE(*,601) N,XN,YN,YN,YN,D,H
601 FORMAT(I6,4F12.6,2E12.4)
    DO WHILE(XN.LT.XEND)
       IF(ABS(XN-XEND) .LT. 1.0E-6) EXIT
       IF(H.LT.0.01*STEP) THEN
         WRITE(*,"(A,E12.6,A)") " H =",H," < 0.01*Initial_Step_Size."
         EXIT
       ENDIF
       XNH = XN + H
       IF(XNH.GT.XEND) THEN
         H=XEND-XN
         XNH=XEND
       ENDIF
       N = N+1
       I = rank +1
          G = GN
          XT = XN+HQ*I
       IF(I<=2) THEN
       G = F(XT,YN+G*HQ*I)
       G = G + F(XT,YN+G*HQ*I)
       ELSE
       DO 21 M=1,PM
          G = F(XT,YN+G*HQ*I)
21     CONTINUE
       ENDIF
       CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
       CALL MPI_ALLGATHER(G,1,MPI_REAL,G0,1,MPI_REAL,MPI_COMM_WORLD,   &
  &        ierr)
       YNH = YN +R6*(GN+2.0*G0(2)+G0(4))*H
!!     HALF STEP-SIZE
       HH = 0.5*H
       YH2 = YN +R6*(GN+2.0*G0(1)+G0(2)+4.0*G0(3)+G0(4))*HH
!!      Predictor and corrector
!!     D = R15*(YH2-YH1)
       D = R15*(2.0*G0(1)+4.0*G0(3)-GN-3.0*G0(2)-G0(4))*HH
       DD = 0.1*DD + ABS(D)
       IF(DD.GT.EPS) THEN
!!       HALF STEP SIZING
         H=HH
         XNH = XN + H
         YNH = YN +R6*(GN+2.0*G0(1)+0.5*G0(2))*HH
       ELSE
!!       DOUBLE STEP SIZING
```

```
          IF(DD.LT.EPS*0.02) H=2*H
        ENDIF
        YNX = YBEGIN/XNH
        IF(rank.EQ.0) WRITE(*,601) N,XNH,YNH,YNX,YH2,D,H
        XN = XNH
        YN = YNH
        GN = F(XN,YN)
        END DO
      CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
      CALL MPI_FINALIZE(ierr)
      RK4S=YN
      RETURN
      END
      REAL FUNCTION FXY(X,Y)
      REAL X,Y
      FXY = 1.0/X**2 - Y/X -Y*Y
!      FXY =  EXP(X)
      RETURN
      END
```

Appendix C. Scalable Implementation

```
! Uniform step size
! Fourth Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                            Y(XBEGIB) = YBEGIN
! to the point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      include "mpif.h"
      INTEGER NSTEPS,IDATA(2),size,rank,tag,ierr
      INTEGER status(MPI_STATUS_SIZE)
      REAL RDATA(3,99),XBGN(99),XEND(99),YBGN(99),YEND(99)
      REAL, EXTERNAL :: FXY
      CALL MPI_INIT(ierr)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
      CALL MPI_TYPE_CONTIGUOUS(3,MPI_REAL,TYPE_R3,ierr)
      CALL MPI_TYPE_COMMIT(TYPE_R3,ierr)
  501 FORMAT(4I5)
  502 FORMAT(3F10.4)
      tag=1
      IF(rank.EQ.0) THEN
        OPEN(10,FILE="rk4data1")
        READ(10,501) ND1,NSTEPS
        READ(10,502)(XBGN(I),XEND(I),YBGN(I),I=1,ND1)
        WRITE(*,'(A,2I8)') "ND1,NSTEPS:",ND1,NSTEPS,"I,XBGN,XEND,YBGN:"
        WRITE(*,'(I6,3F14.6)') (I,XBGN(I),XEND(I),YBGN(I),I=1,ND1)
        WRITE(*,"(A,I4,A,I4)") "Size=",size,"  Rank= ",rank
        LC=ND1/size
        I2=LC
        IF(size.GT.1) THEN
          IDATA(1) = ND1
          IDATA(2) = NSTEPS
          CALL MPI_BCAST(IDATA,2,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
          LCR=MOD(ND1,size)
          IF(LCR.EQ.0) THEN
            I1=LC
          ELSE
            I1=LC+1
          ENDIF
          DO 510 L=1,size-1
             IF(L.LE.LCR) THEN
               I2=LC+1
             ELSE
               I2=LC
             ENDIF
             DO 505 J=1,I2
                RDATA(1,J)=XBGN(I1+J)
                RDATA(2,J)=XEND(I1+J)
                RDATA(3,J)=YBGN(I1+J)
  505        CONTINUE
             CALL MPI_SEND(RDATA,I2,TYPE_R3,L,tag,MPI_COMM_WORLD,ierr)
```

```
 510     CONTINUE
       ENDIF
       I1=0
     ELSE
       CALL MPI_BCAST(IDATA,2,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
       ND1 = IDATA(1)
       NSTEPS = IDATA(2)
       LC=ND1/size
       LCR=MOD(ND1,size)
       IF(rank.LT.LCR) THEN
         I2=LC+1
         I1=rank*I2
       ELSE
         I2=LC
         I1=LCR*(LC+1)+LC*(rank-LCR)
       ENDIF
       CALL MPI_RECV(RDATA,I2,TYPE_R3,0,tag,MPI_COMM_WORLD,            &
    &                status,ierr)
       DO 515 J=1,I2
          XBGN(J)=RDATA(1,J)
          XEND(J)=RDATA(2,J)
          YBGN(J)=RDATA(3,J)
 515   CONTINUE
     ENDIF
     DO 520 I=1,I2
        YEND(I) = RK4S(FXY,NSTEPS,XBGN(I),XEND(I),YBGN(I))
 520 CONTINUE
     tag=99
     IF(rank.GT.0) THEN
!234567891123456789212345678931234567894123456789512345678961234567897123 45
       CALL MPI_SEND(YEND,I2,TYPE_REAL,0,tag,MPI_COMM_WORLD,           &
    &                   ierr)
     ELSE
       I1=1
       DO 530 L=1,size-1
          IF(L.LT.LCR) THEN
            I2=LC+1
          ELSE
            I2=LC
          ENDIF
          I1=I1+I2
          CALL MPI_RECV(YEND(I1),I2,MPI_REAL,L,tag,MPI_COMM_WORLD,     &
    &                status,ierr)
 530   CONTINUE
       WRITE(*,'(A,F10.6,A,E14.6)')                                    &
    &  ("Y(",XEND(I)," ) =",YEND(I),I=1,ND1)
     ENDIF
     CALL MPI_FINALIZE(ierr)
     STOP
     END
     FUNCTION RK4S(F,NSTEPS,XBGN,XEND,YBGN)
     REAL, EXTERNAL :: F
     INTEGER N,NSTEPS
     REAL XBGN,XEND,YBGN,YEND
```

```
      REAL DERIV,H,K1,K2,K3,K4,XN,YN
      WRITE(*,'(A,I4,3F14.6)') "NSTEPS,XBGN,XEND,YBGN:",NSTEPS,XBGN,    &
     &                          XEND,YBGN
      IF(NSTEPS.LT.1) NSTEPS=1
      H = (XEND-XBGN)/NSTEPS
      WRITE(*,'(A,3F14.6)') "H:",H
      XN = XBGN
      YN = YBGN
      DERIV = F(XN,YN)
      R6 = 1.0/6
      N=0
      WRITE(*,"(4X,A,A)") " N         XN           YN         YX",         &
     &                    "       YN-YX       DERIV "
      WRITE(*,601) N,XN,YN,YN,YN-YN,DERIV
  601 FORMAT(I6,3F12.6,3E12.4)
      DO 10 N=1,NSTEPS
         K1 = H*DERIV
         K2 = H*F(XN+0.5*H,YN+0.5*K1)
         K3 = H*F(XN+0.5*H,YN+0.5*K2)
         K4 = H*F(XN+H,YN+K3)
         YN = YN +R6*(K1+2*K2+2*K3+K4)
         XN = XBGN + N*H
         YN0 = 1.0/XN
         DERIV = F(XN,YN)
         WRITE(*,601) N,XN,YN,YN0,YN-YN0,DERIV
   10    CONTINUE
      RK4S=YN
      RETURN
      END
      REAL FUNCTION FXY(X,Y)
      REAL X,Y
      FXY= 1.0/X**2 - Y/X -Y*Y
      RETURN
      END
```

```
! Runge-Kutta Step Control Method
! Runge-Kutta Predictor-Corrector Method
! with Half or Double step sizing at run time.
! Fourth Order Runge-Kutta Method to solve the First Order
! Differential Equation
!              Y'(X) = F(X,Y)
! with Initial Condition of
!                            Y(XBEGIB) = YBEGIN
! to the point XEND.
! A Function subprogram called "F" must be supplied
      PROGRAM MAIN
      include "mpif.h"
      INTEGER IDATA(2),size,rank,tag,ierr
      INTEGER status(MPI_STATUS_SIZE)
      REAL RDATA(4,99),HBGN(99),XBGN(99),XEND(99),YBGN(99),YEND(99)
      REAL, EXTERNAL :: FXY
```

```
      CALL MPI_INIT(ierr)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
      CALL MPI_TYPE_CONTIGUOUS(4,MPI_REAL,TYPE_R4,ierr)
      CALL MPI_TYPE_COMMIT(TYPE_R4,ierr)
  501 FORMAT(I5)
  502 FORMAT(4F10.4)
      tag=1
      IF(rank.EQ.0) THEN
        OPEN(10,FILE="rk4data2")
        READ(10,501) ND1
        READ(10,502)(HBGN(I),XBGN(I),XEND(I),YBGN(I),I=1,ND1)
        WRITE(*,'(A,I6)') "ND1:",ND1,"I,HBGN,XBGN,XEND,YBGN:"
!234567891123456789212345678931234567894123456789512345678961234567897123 45
        WRITE(*,'(I6,4F14.6)') (I,HBGN(I),XBGN(I),XEND(I),YBGN(I),      &
     &                           I=1,ND1)
        WRITE(*,"(A,I4,A,I4)") "Size=",size,"  Rank= ",rank
        LC=ND1/size
        I2=LC
        IF(size.GT.1) THEN
          CALL MPI_BCAST(ND1,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
          LCR=MOD(ND1,size)
          IF(LCR.EQ.0) THEN
            I1=LC
          ELSE
            I1=LC+1
          ENDIF
          DO 510 L=1,size-1
             IF(L.LE.LCR) THEN
               I2=LC+1
             ELSE
               I2=LC
             ENDIF
             DO 505 J=1,I2
                RDATA(1,J)=HBGN(I1+J)
                RDATA(2,J)=XBGN(I1+J)
                RDATA(3,J)=XEND(I1+J)
                RDATA(4,J)=YBGN(I1+J)
  505        CONTINUE
             CALL MPI_SEND(RDATA,I2,TYPE_R4,L,tag,MPI_COMM_WORLD,ierr)
             I1=I1+I2
  510     CONTINUE
        ENDIF
        I1=0
      ELSE
        CALL MPI_BCAST(ND1,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
        LC=ND1/size
        LCR=MOD(ND1,size)
        IF(rank.LT.LCR) THEN
          I2=LC+1
          I1=rank*I2
        ELSE
          I2=LC
          I1=LCR*(LC+1)+LC*(rank-LCR)
```

```
      ENDIF
      CALL MPI_RECV(RDATA,I2,TYPE_R4,0,tag,MPI_COMM_WORLD,            &
     &              status,ierr)
      DO 515 J=1,I2
         HBGN(J)=RDATA(1,J)
         XBGN(J)=RDATA(2,J)
         XEND(J)=RDATA(3,J)
         YBGN(J)=RDATA(4,J)
515   CONTINUE
    ENDIF
    DO 520 I=1,I2
       YEND(I) = RK4SC(FXY,HBGN(I),XBGN(I),XEND(I),YBGN(I))
520 CONTINUE
    tag=99
    IF(rank.GT.0) THEN
      CALL MPI_SEND(YEND,I2,MPI_REAL,0,tag,MPI_COMM_WORLD,            &
     &                  ierr)
    ELSE
      I1=1
      DO 530 L=1,size-1
         IF(L.LT.LCR) THEN
           I2=LC+1
         ELSE
           I2=LC
         ENDIF
         I1=I1+I2
         CALL MPI_RECV(YEND(I1),I2,MPI_REAL,L,tag,MPI_COMM_WORLD,     &
     &              status,ierr)
530   CONTINUE
      WRITE(*,'(A,F10.6,A,E14.6)')                                     &
     &  ("Y(",XEND(I)," ) =",YEND(I),I=1,ND1)
    ENDIF
    CALL MPI_FINALIZE(ierr)
    STOP
    END
    FUNCTION RK4SC(F,STEP,XBEGIN,XEND,YBEGIN)
    REAL, EXTERNAL :: F
    INTEGER N
    REAL DERIV,STEP,H,K1,K2,K3,K4,XBEGIN,XN,XEND,YBEGIN,YN
    REAL EPS,R6,D,DD,HH,YH,YHH
    XN = XBEGIN
    YN = YBEGIN
    DERIV = F(XN,YN)
    EPS=1.0E-6
    R6 = 1.0/6
    R15 = 1.0/15.0
    H = STEP
    DD = 0.1*EPS
    D = 0.0
    N=0
    WRITE(*,"(4X,A,A)") " N         XN          YN          YNX",      &
     &                    "           D          H"
    WRITE(*,601) N,XN,YN,YN,D,H
601 FORMAT(I6,3F12.6,E14.4,F10.4)
```

```
      DO WHILE(XN.LE.XEND)
         IF(ABS(XN-XEND) .LT. 1.0E-6) EXIT
         XNH = XN + H
         IF(XNH.GT.XEND) THEN
           H=XEND-XN
           XNH=XEND
         ENDIF
         N = N+1
         K1 = H*DERIV
         K2 = H*F(XN+0.5*H,YN+0.5*K1)
         K3 = H*F(XN+0.5*H,YN+0.5*K2)
         K4 = H*F(XN+H,YN+K3)
         YH = YN +R6*(K1+2*K2+2*K3+K4)
!!       HALF STEP-SIZE
         HH = 0.5*H
         K1 = 0.5*K1
         K2 = HH*F(XN+0.5*HH,YN+0.5*K1)
         K3 = HH*F(XN+0.5*HH,YN+0.5*K2)
         K4 = HH*F(XN+HH,YN+K3)
         YH1 = YN +R6*(K1+2*K2+2*K3+K4)
         K1 = HH*F(XN+HH,YH1)
         K2 = HH*F(XN+1.5*HH,YH1+0.5*K1)
         K3 = HH*F(XN+1.5*HH,YH1+0.5*K2)
         K4 = HH*F(XN+2.0*HH,YH1+K3)
!!        Predictor and corrector
         YHH = YH1 +R6*(K1+2*K2+2*K3+K4)
         D = R15*(YHH-YH)
         YNH = YHH + D
         DD = 0.1*DD + ABS(D)
         IF(DD.GT.EPS) THEN
!!         HALF STEP SIZING
           H=HH
           XNH = XN + H
           YNH=YH1
         ELSE
!!         DOUBLE STEP SIZING
           IF(DD.LT.EPS*0.02) H=2*H
         ENDIF
         YNX=1.0/XNH
         WRITE(*,601) N,XNH,YNH,YNX,D,H
         XN = XNH
         YN = YNH
         DERIV = F(XN,YN)
         END DO
      RK4S=YN
      RETURN
      END
      REAL FUNCTION FXY(X,Y)
      REAL X,Y
      FXY = 1.0/X**2 - Y/X -Y*Y
!     FXY =  EXP(X)
      RETURN
      END
```