THE NAS PARALLEL BENCHMARKS

D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga
RNR Technical Report RNR-94-007, March 1994

Abstract

A new set of benchmarks has been developed for the performance evaluation of highly parallel supercomputers. These benchmarks consist of five parallel kernels and three simulated application benchmarks. Together they mimic the computation and data movement characteristics of large scale computational fluid dynamics (CFD) applications.

The principal distinguishing feature of these benchmarks is their "pencil and paper" specification—all details of these benchmarks are specified only algorithmically. In this way many of the difficulties associated with conventional benchmarking approaches on highly parallel systems are avoided.

This paper is an updated version of a RNR technical report originally issued January 1991. Bailey, Barszcz, Barton, Carter, and Lasinski are with NASA Ames Research Center. Dagum, Fatoohi, Fineberg, Simon and Weeratunga are with Computer Sciences Corp. (work supported through NASA Contract NAS 2-12961). Schreiber is with RIACS (work funded via NASA cooperative agreement NCC 2-387). Browning is now in the Netherlands (work originally done with CSC). Frederickson is now with Cray Research, Inc. in Los Alamos, NM (work originally done with RIACS). Venkatakrishnan is now with ICASE in Hampton, VA (work originally done with CSC). Address for all authors except Browning, Frederickson and Venkatakrishnan: NASA Ames Research Center, Mail Stop T27A-1, Moffett Field, CA 94035-1000.

1 GENERAL REMARKS

D. Bailey, D. Browning,* R. Carter, S. Fineberg,* and H. Simon*

1.1 Introduction

The Numerical Aerodynamic Simulation (NAS) Program, which is based at NASA Ames Research Center, is a large scale effort to advance the state of computational aerodynamics. Specifically, the NAS organization aims "to provide the Nation's aerospace research and development community by the year 2000 a high-performance, operational computing system capable of simulating an entire aerospace vehicle system within a computing time of one to several hours" (ref. 1, p. 3). The successful solution of this "grand challenge" problem will require the development of computer systems that can perform the required complex scientific computations at a sustained rate nearly one thousand times greater than current generation supercomputers can now achieve. The architecture of computer systems able to achieve this level of performance will likely be dissimilar to the shared memory multiprocessing supercomputers of today. While no consensus yet exists on what the design will be, it is likely that the system will consist of at least 1000 processors computing in parallel.

Highly parallel systems with computing power roughly equivalent to traditional shared memory multiprocessors exist today. Unfortunately, the performance evaluation of these systems on comparable types of scientific computations is very difficult for several reasons. Few relevant data are available for the performance of algorithms of interest to the computational aerophysics community on many currently available parallel systems. Benchmarking and performance evaluation of such systems has not kept pace with advances in hardware, software and algorithms. In particular, there is as yet no generally accepted benchmark program or even a benchmark strategy for these systems.

The popular "kernel" benchmarks that have been used for traditional vector supercomputers, such as the Livermore Loops [2], the LINPACK benchmark [3, 4] and the original NAS Kernels [5], are clearly inappropriate for

*Computer Sciences Corporation, El Segundo, California. This work is supported through NASA Contract NAS 2-12961.

the performance evaluation of highly parallel machines. First of all, the tuning restrictions of these benchmarks rule out many widely used parallel extensions. More importantly, the computation and memory requirements of these programs do not do justice to the vastly increased capabilities of the new parallel machines, particularly those systems that will be available by the mid-1990s.

On the other hand, a full scale scientific application is similarly unsuitable. First of all, porting a large program to a new parallel computer architecture requires a major effort, and it is usually hard to justify a major research task simply to obtain a benchmark number. For that reason we believe that the otherwise very successful PERFECT Club benchmark [6] is not suitable for highly parallel systems. This is demonstrated by very sparse performance results for parallel machines in the recent reports [7, 8, 9].

Alternatively, an application benchmark could assume the availability of automatic software tools for transforming "dusty deck" source into efficient parallel code on a variety of systems. However, such tools do not exist today, and many scientists doubt that they will ever exist across a wide range of architectures.

Some other considerations for the development of a meaningful benchmark for a highly parallel supercomputer are the following:

- Advanced parallel systems frequently require new algorithmic and software approaches, and these new methods are often quite different from the conventional methods implemented in source code for a sequential or vector machine.

- Benchmarks must be "generic" and should not favor any particular parallel architecture. This requirement precludes the usage of any architecture-specific code, such as message passing code.

- The correctness of results and performance figures must be easily verifiable. This requirement implies that both input and output data sets must be kept very small. It also implies that the nature of the computation and the expected results must be specified in great detail.

- The memory size and run time requirements must be easily adjustable to accommodate new systems with increased power.

- The benchmark must be readily distributable.

In our view, the only benchmarking approach that satisfies all of these constraints is a "paper and pencil" benchmark. The idea is to specify a set of problems only algorithmically. Even the input data must be specified only on paper. Naturally, the problem has to be specified in sufficient detail that a unique solution exists, and the required output has to be brief yet detailed enough to certify that the problem has been solved correctly. The person or persons implementing the benchmarks on a given system are expected to solve the various problems in the most appropriate way for the specific system. The choice of data structures, algorithms, processor allocation and memory usage are all (to the extent allowed by the specification) left open to the discretion of the implementer. Some extension of Fortran or C is required, and reasonable limits are placed on the usage of assembly code and the like, but otherwise programmers are free to utilize language constructs that give the best performance possible on the particular system being studied.

To this end, we have devised a number of relatively simple "kernels," which are specified completely in chapter 2 of this document. However, kernels alone are insufficient to completely assess the performance potential of a parallel machine on real scientific applications. The chief difficulty is that a certain data structure may be very efficient on a certain system for one of the isolated kernels, and yet this data structure would be inappropriate if incorporated into a larger application. In other words, the performance of a real computational fluid dynamics (CFD) application on a parallel system is critically dependent on data motion between computational kernels. Thus we consider the complete reproduction of this data movement to be of critical importance in a benchmark.

Our benchmark set therefore consists of two major components: five parallel kernel benchmarks and three simulated application benchmarks. The simulated application benchmarks combine several computations in a manner that resembles the actual order of execution in certain important CFD application codes. This is discussed in more detail in chapter 3.

We feel that this benchmark set successfully addresses many of the problems associated with benchmarking parallel machines. Although we do not claim that this set is typical of all scientific computing, it is based on the key components of several large aeroscience applications used on supercomputers by scientists at NASA Ames Research Center. These benchmarks will be used by the Numerical Aerodynamic Simulation (NAS) Program to evaluate the performance of parallel computers.

1.2 Benchmark Rules

1.2.1 Definitions

In the following, the term "processor" is defined as a hardware unit capable of executing both floating point addition and floating point multiplication instructions. The "local memory" of a processor refers to randomly accessible memory that can be accessed by that processor in less than one microsecond. The term "main memory" refers to the combined local memory of all processors. This includes any memory shared by all processors that can be accessed by each processor in less than one microsecond. The term "mass storage" refers to non-volatile randomly accessible storage media that can be accessed by at least one processor within forty milliseconds. A "processing node" is defined as a hardware unit consisting of one or more processors plus their local memory, which is logically a single unit on the network that connects the processors.

The term "computational nodes" refers to those processing nodes primarily devoted to high-speed floating point computation. The term "service nodes" refers to those processing nodes primarily devoted to system operations, including compilation, linking and communication with external computers over a network.

1.2.2 General rules

Implementations of these benchmarks must be based on either Fortran-90 (which includes Fortran-77 as a subset) or C, although a wide variety of parallel extensions are allowed. This requirement stems from the observation that Fortran and C are the most commonly used programming languages by the scientific parallel computing community at the present time. If in the future, other languages gain wide acceptance in this community, they will be considered for inclusion in this group. Assembly language and other low-level languages and constructs may not be used, except that certain specific vendor-supported assembly-coded library routines may be called (see section 1.2.3).

We are of the opinion that such language restrictions are necessary, because otherwise considerable effort would be made by benchmarkers in low-level or assembly-level coding. Then the benchmark results would tend to reflect the amount of programming resources available to the benchmarking organization, rather than the fundamental merits of the parallel system. Certainly the mainstream scientists that these parallel computers are intended to serve will be coding applications at the source level, almost certainly in Fortran or C, and thus these benchmarks are designed to measure the performance that can be expected from such code.

Accordingly, the following rules must be observed in any implementations of the NAS Parallel Benchmarks:

- All floating point operations must be performed using 64-bit floating point arithmetic.
- All benchmarks must be coded in either Fortran-90 [10] or C [11], with certain approved extensions.
- Implementation of the benchmarks may not include a mix of Fortran-90 and C code—one or the other must be used.
- Any extension of Fortran-90 that is in the High Performance Fortran (HPF) draft dated January 1992 or later [12] is allowed.
- Any language extension or library routine that is employed in any of the benchmarks must be supported by the vendor and available to all users.
- Subprograms and library routines not written in Fortran or C may only perform certain functions, as indicated in the next section.
- All rules apply equally to subroutine calls, language extensions and compiler directives (i.e., special comments).

1.2.3 Allowable Fortran extensions and library routines

Fortran extensions and library routines are also permitted that perform the following:

- Indicate sections of code that can be executed in parallel or loops that can be distributed among different computational nodes.
- Specify the allocation and organization of data among or within computational nodes.

- Communicate data between processing nodes.
- Communicate data between the computational nodes and service nodes.
- Rearrange data stored in multiple computational nodes, including constructs to perform indirect addressing and array transpositions.
- Synchronize the action of different computational nodes.
- Initialize for a data communication or synchronization operation that will be performed or completed later.
- Perform high-speed input or output operations between main memory and the mass storage system.
- Perform any of the following array reduction operations on an array either residing within a single computational node or distributed among multiple nodes: $+, \times$, `MAX, MIN, AND, OR, XOR`.
- Combine communication between nodes with one of the operations listed in the previous item.
- Perform any of the following computational operations on arrays either residing within a single computational node or distributed among multiple nodes: dense or sparse matrix-matrix multiplication, dense or sparse matrix-vector multiplication, one-dimensional, two-dimensional or three-dimensional fast Fourier transforms, sorting, block tri-diagonal system solution and block penta-diagonal system solution. Such routines must be callable with general array dimensions.

1.3 Sample Codes

The intent of this paper is to completely specify the computation to be carried out. Theoretically, a complete implementation, including the generation of the correct input data, could be produced from the information in this paper. However, the developers of these benchmarks are aware of the difficulty and time required to generate a correct implementation from scratch in this manner. Furthermore, despite several reviews, ambiguities in this technical paper may exist that could delay implementations.

In order to reduce the number of difficulties and to aid the benchmarking specialist, Fortran-77 computer programs implementing the benchmarks are available. These codes are to be considered examples of how the problems could be solved on a single processor system, rather than statements of how they should be solved on an advanced parallel system. The sample codes actually solve scaled down versions of the benchmarks that can be run on many current generation workstations. Instructions are supplied in comments in the source code on how to scale up the program parameters to the full size benchmark specifications.

These programs, as well as the benchmark document itself, are available from the Systems Development Branch in the NAS Systems Division. Mail Stop 258-5, NASA Ames Research Center, Moffett Field, CA 94035-1000, attn: NAS Parallel Benchmark Codes. The sample codes are provided on Macintosh floppy disks and contain the Fortran source codes, "ReadMe" files, input data files, and reference output data files for correct implementations of the benchmark problems. These codes have been validated on a number of computer systems ranging from conventional workstations to supercomputers.

Three classes of problems are defined in this document. These will be denoted "Sample Code," "Class A," and "Class B," since the three classes differ mainly in the sizes of principal arrays. Tables 1.1, 1.2, and 1.3 give the problem sizes, memory requirements (measured in Mw), run times and performance rates (measured in Mflop/s) for each of the eight benchmarks and for the Sample Code, Class A, and Class B problem sets. These statistics are based on implementations on one processor of a Cray Y-MP. The operation count for the Integer Sort benchmark is based on integer operations rather than floating-point operations. The entries in the "Problem Size" columns are sizes of key problem parameters. Complete descriptions of these parameters are given in chapters 2 and 3.

1.4 Submission of Benchmark Results

It must be emphasized that the sample codes described in section 1.3 are not the benchmark codes, but only implementation aids. For the actual benchmarks, the sample codes must be scaled to larger problem sizes. The sizes of the current benchmarks were chosen so that implementations are possible on currently available supercomputers. As parallel computer technology

Benchmark code	Problem size	Memory (Mw)	Time (sec)	Rate (Mflop/s)
Embarrassingly parallel (EP)	2^{24}	0.1	11.6	120
Multigrid (MG)	32^3	0.1	0.1	128
Conjugate gradient (CG)	1400	1.0	1.2	63
3-D FFT PDE (FT)	64^3	2.0	1.2	160
Integer sort (IS)	2^{16}	0.3	0.2	30.5
LU solver (LU)	12^3	0.3	3.5	28
Pentadiagonal solver (SP)	12^3	0.2	7.2	24
Block tridiagonal solver (BT)	12^3	0.3	7.2	34

Table 1.1: NAS Parallel Benchmarks Sample Code Statistics

progresses, future releases of these benchmarks will specify larger problem sizes.

The authors and developers of these benchmarks encourage submission of performance results for the problems listed in table 1.2. Periodic publication of the submitted results is planned. Benchmark results should be submitted to the Applied Research Branch, NAS Systems Division, Mail Stop T045-1, NASA Ames Research Center, Moffett Field, CA 94035, attn: NAS Parallel Benchmark Results. A complete submission of results should include the following:

- A detailed description of the hardware and software configuration used for the benchmark runs.

- A description of the implementation and algorithmic techniques used.

- Source listings of the benchmark codes.

- Output listings from the benchmarks.

Benchmark code	Problem size	Memory (Mw)	Time (sec)	Rate (Mflop/s)
Embarrassingly parallel (EP)	2^{28}	1	151	147
Multigrid (MG)	256^3	57	54	154
Conjugate gradient (CG)	14000	10	22	70
3-D FFT PDE (FT)	$256^2 \times 128$	59	39	192
Integer sort (IS)	2^{23}	26	21	37.2
LU solver (LU)	64^3	30	344	189
Pentadiagonal solver (SP)	64^3	6	806	175
Block tridiagonal solver (BT)	64^3	24	923	192

Table 1.2: NAS Parallel Benchmarks Class A Statistics

Benchmark code	Problem size	Memory (Mw)	Time (sec)	Rate (Mflop/s)
Embarrassingly parallel (EP)	2^{30}	18	512	197
Multigrid (MG)	256^3	59	114	165
Conjugate gradient (CG)	75000	97	998	55
3-D FFT PDE (FT)	$512 \times 256 \times 256$	162	366	195
Integer sort (IS)	2^{25}	114	126	25
LU solver (LU)	102^3	122	1973	162
Pentadiagonal solver (SP)	102^3	22	2160	207
Block tridiagonal solver (BT)	102^3	96	3554	203

Table 1.3: NAS Parallel Benchmarks Class B Statistics

References

[1] Numerical Aerodynamic Simulation Program Plan. NAS Systems Division, Ames Research Center, October 1988.

[2] McMahon, F. H.: The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range. Technical Report UCRL-53745, Lawrence Livermore National Laboratory, Livermore, Calif., Dec. 1986.

[3] Dongarra, J. J.: The LINPACK Benchmark: An Explanation. SuperComputing, Spring 1988, pp. 10-14.

[4] Dongarra, J. J.: Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment. TR MCSRD 23, Argonne National Laboratory, March 1988.

[5] Bailey, D. H.; and Barton, J.: The NAS Kernel Benchmark Program. Technical Report 86711, Ames Research Center, Moffett Field, Calif., August 1985.

[6] Berry, M.; et al. The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers. The International Journal of Supercomputer Applications, vol. 3, 1989, pp. 5-40.

[7] Pointer, L.: PERFECT Report 1. Technical Report 896, CSRD, Univ. of Illinois, Urbana, Ill., July 1989.

[8] Pointer, L.: PERFECT Report 2: Performance Evaluation for Costeffective Transformations. Technical Report 964, CSRD, Univ. of Illinois, Urbana, Ill., March 1990.

[9] Cybenko, G.; Kipp, L.; Pointer, L.; and Kuck, D.: Supercomputer Performance Evaluation and the Perfect Benchmarks. Technical Report 965, CSRD, Univ. of Illinois, Urbana, Ill., March 1990.

[10] American National Standard Programming Language Fortran X3.198-1992. American National Standards Institute, 1430 Broadway, New York, NY 10018, 1992.

[11] Draft Proposed C ANSI Standard X3J3–S8115. American National Standards Institute, 1430 Broadway, New York, NY, 1990.

[12] High Performance Fortran (HPF) Language Specification, Version 1.0 Draft. Center for Research in Parallel Computing, P.O. 1892, Rice University, Houston, TX 77251, January 1993.

2 THE KERNEL BENCHMARKS

D. Bailey, E. Barszcz, L. Dagum,* P. Frederickson,† R. Schreiber,† and H. Simon*

2.1 Overview

After an evaluation of a number of large scale CFD and computational aerosciences applications on the NAS supercomputers at NASA Ames, a number of kernels were selected for the benchmark. These were supplemented by some other kernels which are intended to test specific features of parallel machines. The following benchmark set was then assembled:

EP: An "embarrassingly parallel" kernel. It provides an estimate of the upper achievable limits for floating point performance, i.e., the performance without significant interprocessor communication.

MG: A simplified multigrid kernel. It requires highly structured long distance communication and tests both short and long distance data communication.

CG: A conjugate gradient method is used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. This kernel is typical of unstructured grid computations in that it tests irregular long distance communication, employing unstructured matrix vector multiplication.

FT: A 3-D partial differential equation solution using FFTs. This kernel performs the essence of many "spectral" codes. It is a rigorous test of long-distance communication performance.

IS: A large integer sort. This kernel performs a sorting operation that is important in "particle method" codes. It tests both integer computation speed and communication performance.

*Computer Sciences Corporation. This work is supported through NASA Contract NAS 2-12961.

†Research Institute for Advanced Computer Science (RIACS), Ames Research Center. This work is supported by NAS Systems Division through Cooperative Agreement Number NCC 2-387.

These kernels involve substantially larger computations than previous kernel benchmarks, such as the Livermore Loops or Linpack, and therefore they are more appropriate for the evaluation of parallel machines. The Parallel Kernels in particular are sufficiently simple that they can be implemented on a new system without unreasonable effort and delay. Most importantly, as emphasized earlier, this set of benchmarks incorporates a new concept in performance evaluation, namely that only the computational task is specified, and that the actual implementation of the kernel can be tailored to the specific architecture of the parallel machine.

In this chapter the Parallel Kernel benchmarks are presented, and the particular rules for allowable changes are discussed. Future reports will describe implementations and benchmarking results on a number of parallel supercomputers.

2.2 Description of the Kernels

2.2.1 Kernel EP: An embarrassingly parallel benchmark

D. Bailey and P. Frederickson

Brief Statement of Problem

Generate pairs of Gaussian random deviates according to a specific scheme described below and tabulate the number of pairs in successive square annuli.

Detailed Description

Set $n = 2^{28}$, $a = 5^{13}$ and $s = 271,828,183$. Generate the pseudorandom floating point values r_j in the interval (0, 1) for $1 \leq j \leq 2n$ using the scheme described in section 2.3. Then for $1 \leq j \leq n$ set $x_j = 2r_{2j-1} - 1$ and $y_j = 2r_{2j} - 1$. Thus x_j and y_j are uniformly distributed on the interval $(-1, 1)$.

Next set $k = 0$. Then beginning with $j = 1$, test to see if $t_j = x_j^2 + y_j^2 \leq 1$. If not, reject this pair and proceed to the next j. If this inequality holds, then set $k \leftarrow k+1$, $X_k = x_j\sqrt{(-2\log t_j)/t_j}$ and $Y_k = y_j\sqrt{(-2\log t_j)/t_j}$, where log denotes the natural logarithm. Then X_k and Y_k are independent Gaussian deviates with mean zero and variance one. Approximately $n\pi/4$ pairs will be constructed in this manner. See reference 2, page 117 for additional discussion of this scheme for generating Gaussian deviates. The computation of Gaussian deviates must employ the above formula, and vendor-supplied

l	Class A Q_l	Class B Q_l
0	98257395	393058470
1	93827014	375280898
2	17611549	70460742
3	1110028	4438852
4	26536	105691
5	245	948
6	0	5
7	0	0
8	0	0
9	0	0

Table 2.1: EP Benchmark Counts

intrinsic routines must be employed for performing all square root and logarithm evaluations.

Finally, for $0 \leq l \leq 9$ tabulate Q_l as the count of the pairs (X_k, Y_k) that lie in the square annulus $l \leq \max(|X_k|, |Y_k|) < l + 1$, and output the ten Q_l counts. The two sums $\sum_k X_k$ and $\sum_k Y_k$ must also be output.

This completes the definition of the Class A problem. The Class B problem is the same except that $n = 2^{30}$.

Verification Test

The two sums $\sum_k X_k$ and $\sum_k Y_k$ must agree with reference values to within one part in 10^{12}. For the Class A problemm, the reference values are $-4.295875165629892 \times 10^3$ and $-1.580732573678431 \times 10^4$, while for the Class B problem the sums are $4.033815542441498 \times 10^4$ and $-2.660669192809235 \times 10^4$. Each of the ten Q_l counts must agree exactly with reference values, which are given in table 2.1.

Operations to be Timed

All of the operations described above are to be timed, including tabulation and output.

Other Features

- This problem is typical of many Monte Carlo simulation applications.
- The only requirement for communication is the combination of the 10 sums from various processors at the end.
- Separate sections of the uniform pseudorandom numbers can be independently computed on separate processors. See section 2.3 for details.
- The smallest distance between a floating-point value and a nearby integer among the r_j, X_k and Y_k values is 3.2×10^{-11}, which is well above the achievable accuracy using 64 bit floating arithmetic on existing computer systems. Thus if a truncation discrepancy occurs, it implies a problem with the system hardware or software.

2.2.2 Kernel MG: a simple 3D multigrid benchmark

E. Barszcz and P. Frederickson

Brief Statement of Problem

Four iterations of the V-cycle multigrid algorithm described below are used to obtain an approximate solution u to the discrete Poisson problem

$$\nabla^2 u = v$$

on a $256 \times 256 \times 256$ grid with periodic boundary conditions.

Detailed Description

Set $v = 0$ except at the twenty points listed in table 2.2. where $v = \pm 1$. (These points were determined as the locations of the ten largest and ten smallest pseudorandom numbers generated as in Kernel FT.)

Begin the iterative solution with $u = 0$. Each of the four iterations consists of the following two steps, in which $k = 8 = \log_2(256)$:

$$r = v - A\,u \qquad (\textit{evaluate residual})$$

$$u = u + M^k\,r \qquad (\textit{apply correction})$$

Here M^k denotes the V-cycle multigrid operator, defined in table 2.3.

$v_{i,j,k}$	(i,j,k)				
–1.0	211,154, 98	102,138,112	101,156, 59	17,205, 32	92, 63,205
	199, 7,203	250,170,157	82,184,255	154,162, 36	223, 42,240
+1.0	57,120,167	5,118,175	176,246,164	45,194,234	212, 7,248
	115,123,207	202, 83,209	203, 18,198	243,172, 14	54,209, 40

Table 2.2: Nonzero values for v

$$
\begin{array}{llllll}
z_k & = & M^k r_k & : & & \\
& \text{if } k > 1 & & & & \\
& & r_{k-1} & = & P\ r_k & \text{(restrict residual)} \\
& & z_{k-1} & = & M^{k-1} r_{k-1} & \text{(recursive solve)} \\
& & z_k & = & Q\ z_{k-1} & \text{(prolongate)} \\
& & r_k & = & r_k\ -\ A\ z_k & \text{(evaluate residual)} \\
& & z_k & = & z_k\ +\ S\ r_k & \text{(apply smoother)} \\
& \text{else} & & & & \\
& & z_1 & = & S\ r_1. & \text{(apply smoother)}
\end{array}
$$

Table 2.3: V-cycle multigrid operator

In this definition A denotes the trilinear finite element discretization of the Laplacian ∇^2 normalized as indicated in table 2.4, where the coefficients of P, Q, and S are also listed.

In this table c_0 denotes the central coefficient of the 27-point operator, when these coefficients are arranged as a $3 \times 3 \times 3$ cube. Thus c_0 is the coefficient that multiplies the value at the gridpoint (i,j,k), while c_1 multiplies the six values at grid points which differ by one in exactly one index, c_2 multiplies the next closest twelve values, those that differ by one in exactly two indices, and c_3 multiplies the eight values located at grid points that differ by one in all three indices. The restriction operator P given in this table is the trilinear projection operator of finite element theory, normalized so that the coefficients of all operators are independent of level, and is half the transpose of the trilinear interpolation operator Q.

C	c_0	c_1	c_2	c_3
A	$-8.0/3.0$	0.0	1.0/6.0	1.0/12.0
P	1.0/2.0	1.0/4.0	1.0/8.0	1.0/16.0
Q	1.0	1.0/2.0	1.0/4.0	1.0/8.0
S(a)	$-3.0/8.0$	$+1.0/32.0$	$-1.0/64.0$	0.0
S(b)	$-3.0/17.0$	$+1.0/33.0$	$-1.0/61.0$	0.0

Table 2.4: Coefficients for trilinear finite element discretization

Verification Test

Class A: Evaluate the residual after four iterations of the V-cycle multi-grid algorithm using the coefficients from the S(a) row for the smoothing operator S, and verify that its L_2 norm

$$\|r\|_2 \ = \ [\ (\ \sum_{i,j,k} r_{i,j,k}\ )/256^3\ ]^{1/2}$$

agrees with the reference value

$$0.2433365309 \times 10^{-05}$$

within an absolute tolerance of 10^{-14}.

Class B: The array size is the same as for Class A (256), but 20 iterations must be performed using the coefficients from the S(b) row for the smoothing operator S. The output $\mathbf{L_2}$ norms must agree with the reference value

$$0.180056440132 \times 10^{-05}$$

within an absolute tolerance of 10^{-14}.

Timing

Start the clock before evaluating the residual for the first time, and after initializing u and v. Stop the clock after evaluating the norm of the final residual, but before displaying or printing its value.

2.2.3 Kernel CG: Solving an unstructured sparse linear system by the conjugate gradient method

R. Schreiber, H. Simon, and R. Carter

Brief Statement of Problem

This benchmark uses the inverse power method to find an estimate of the largest eigenvalue of a symmetric positive definite sparse matrix with a random pattern of nonzeros.

Detailed Description

In the following, A denotes the sparse matrix of order n, lower case Roman letters are vectors, x_j is the j^{th} component of the vector x, and superscript "T" indicates the usual transpose operator. Lower case Greek letters are scalars. We denote by $||x||$ the Euclidean norm of a vector x, $||x|| = \sqrt{x^T x}$. All quantities are real. The inverse power method is to be implemented as follows:

$x = [1, 1, \ldots, 1]^T$;
(start timing here)
DO $it = 1$, $niter$
    Solve the system $Az = x$ and return $||r||$, as described below
    $\zeta = \lambda + 1/(x^T z)$
    Print it, $||r||$, and ζ
    $x = z/||z||$
ENDDO
(stop timing here)

Values for the size of the system n, number of outer iterations $niter$, and the shift λ for three different problem sizes are provided in table 2.5. The solution z to the linear system of equations $Az = x$ is to be approximated using the conjugate gradient (CG) method. This method is to be implemented as follows:

$z = 0$

$r = x$
$\rho = r^T r$
$p = r$
DO $i = 1, 25$
    $q = Ap$
    $\alpha = \rho/(p^T q)$
    $z = z + \alpha p$
    $\rho_0 = \rho$
    $r = r - \alpha q$
    $\rho = r^T r$
    $\beta = \rho/\rho_0$
    $p = r + \beta p$
ENDDO
compute residual norm explicitly: $||r|| = ||x - Az||$

Size	n	$niter$	NONZER	λ
Sample	1400	15	7	10
Class A	14000	15	11	20
Class B	75000	75	13	60

Table 2.5: Input parameters for CG benchmark

Verification Test

The program should print, at every outer iteration of the power method, the iteration number it, the eigenvalue estimate ζ, and the Euclidean norm $||r||$ of the residual vector at the last CG iteration (the vector r in the discussion of CG above). For each size problem the computer value of ζ must agree with the reference value ζ_{REF} within a tolerance of 1.0×10^{-10}, i.e., $|\zeta - \zeta_{REF}| \leq 1.0 \times 10^{-10}$. These reference values ζ_{REF} are provided in table 2.6.

Timing

The reported time must be the wall-clock time required to compute all $niter$ iterations and print the results, after the matrix is generated and downloaded into the system, and after the initialization of the starting vector x.

Size	Computed nonzeros	FLOP $\times 10^9$	mem (MW)	ζ_{REF}
Sample	78148	0.066	1.0	8.59717750786234
Class A	1853104	1.50	10.0	17.13023505380784
Class B	13708072	54.9	96.7	22.712745482078

Table 2.6: Output parameters for CG benchmark

It is permissible initially to reorganize the sparse matrix data structure (*arow*, *acol*, *aelt*), which is produced by the matrix generation routine (described below), to a data structure better suited for the target machine. The original or the reorganized sparse matrix data structure can then be subsequently used in the conjugate gradient interation. Time spent in the initial reorganization of the data structure will *not* be counted towards the benchmark time.

It is also permissible to use several different data structures for the matrix A, keep multiple copies of the matrix A, or to write A to mass storage and read it back in. However, the time for any data movements, which take place within the power iterations (outer iteration) or within the conjugate gradient iterations (inner iteration), must be included in the reported time.

However, the matrix A must be used explicitly. By saving the random sparse vectors x used in **makea** (see below), it is possible to reformulate the sparse matrix-vector multiply operation in such a way that communication is substantially reduced (to only a few dense vectors), and sparse operations are restricted to the processing nodes. Although this scheme of matrix-vector multiplication technically satisfied the original rules of the CG benchmark, it defeats this benchmark's intended purpose of measuring random communication performance. Therefore this scheme is no longer allowed, and results employing implicit matrix-vector multiplication based on the outer product representation of the sparse matrix are no longer considered to be valid benchmark results.

Other Features

The input sparse matrix A is generated by a Fortran 77 subroutine called **makea**, which is provided on the sample code disk described in section 1.3.

In this program, the random number generator is initialized with $a = 5^{13}$ and $s = 314159265$. Then the subroutine **makea** is called to generate the matrix A. This program may not be changed. In the **makea** subroutine the matrix A is represented by the following Fortran 77 variables:

`N (INTEGER)`—the number of rows and columns

`NZ (INTEGER)`—the number of nonzeros

`A (REAL*8)`—array of NZ nonzeros

`IA (INTEGER)`—array of NZ row indices. Element A(K) is in row IA(K) for all $1 \leq$ K $\leq$ NZ.

`JA (INTEGER)`—array of N+1 pointers to the beginnings of columns. Column J of the matrix is stored in positions JA(J) through JA(J+1)-1 of A and IA. JA(N+1) contains NZ+1.

The code generates the matrix as the weighted sum of N outer products of random sparse vectors x:

$$A = \sum_{i=1}^{N} \omega_i x x^T$$

where the weights ω_i are a geometric sequence with $\omega_1 = 1$ and the ratio chosen so that $\omega_N = 0.1$. The vectors x are chosen to have a few randomly placed nonzeros, each of which is a sample from the uniform distribution on (0, 1). Furthermore, the i^{th} element of x_i is set to 1/2 to insure that A cannot be structurally singular. Finally, 0.1 is added to the diagonal of A. This results in a matrix whose condition number (the ratio of its largest eigenvalue to its smallest) is roughly 10. The number of randomly chosen elements of x is provided for each problem size in table 2.5, in the "NONZER" column. The final number of nonzeros of A are listed in table 2.6 in the "computed nonzeros" column. As implemented in the sample codes, the shift λ of the main diagonal of A is the final task in subroutine **makea**. Values are provided for λ in table 2.5.

The data structures used are these. First, a list of triples ($arow$, $acol$, $aelt$) is constructed. Each of these represents an element in row $i = arow$, column $j = acol$, with value $a_{ij} = aelt$. When the $arow$ and $acol$ entries of two of these triples coincide, then the values in their $aelt$ fields are added together in creating a_{ij}. The process of assembling the matrix data structures from the list of triples, including the process of adding coincident entries, is done by the subroutine **sparse**, which is called by **makea** and is also provided. For examples and more details on this sparse data structure, consult section 2.7 of the book by Duff, Erisman, and Reid [3].

2.2.4 Kernel FT: A 3-D fast-Fourier transform partial differential equation benchmark

D. Bailey and P. Frederickson

Brief Statement of Problem

Numerically solve a certain partial differential equation (PDE) using forward and inverse FFTs.

Detailed Description

Consider the PDE

$$\frac{\partial u(x,t)}{\partial t} = \alpha \nabla^2 u(x,t)$$

where x is a position in three-dimensional space. When a Fourier transform is applied to each side, this equation becomes

$$\frac{\partial v(z,t)}{\partial t} = -4\alpha\pi^2 |z|^2 v(z,t)$$

where $v(z,t)$ is the Fourier transform of $u(x,t)$. This has the solution

$$v(z,t) = e^{-4\alpha\pi^2|z|^2 t} v(z,0)$$

Now consider the discrete version of the original PDE. Following the above steps, it can be solved by computing the forward 3-D discrete Fourier transform (DFT) of the original state array $u(x,0)$, multiplying the results by certain exponentials, and then performing an inverse 3-D DFT. The forward DFT and inverse DFT of the $n_1 \times n_2 \times n_3$ array u are defined respectively as

$$F_{q,r,s}(u) = \sum_{l=0}^{n_3-1} \sum_{k=0}^{n_2-1} \sum_{j=0}^{n_1-1} u_{j,k,l} e^{-2\pi ijq/n_1} e^{-2\pi ikr/n_2} e^{-2\pi ils/n_3}$$

$$F^{-1}_{q,r,s}(u) = \frac{1}{n_1 n_2 n_3} \sum_{l=0}^{n_3-1} \sum_{k=0}^{n_2-1} \sum_{j=0}^{n_1-1} u_{j,k,l} e^{2\pi ijq/n_1} e^{2\pi ikr/n_2} e^{2\pi ils/n_3}$$

The specific problem to be solved for the Class A benchmark is as follows. Set $n_1 = 256$, $n_2 = 256$, and $n_3 = 128$. Generate $2n_1n_2n_3$ 64-bit pseudorandom floating point values using the pseudorandom number generator in section 2.3, starting with the initial seed 314159265. Then fill the complex array $U_{j,k,l}$, $0 \le j < n_1$, $0 \le k < n_2$, $0 \le l < n_3$, with this data, where the first dimension varies most rapidly as in the ordering of a 3-D Fortran array. A single complex number entry of U consists of two consecutive pseudorandomly generated results. Compute the forward 3-D DFT of U, using a 3-D fast Fourier transform (FFT) routine, and call the result V. Set $\alpha = 10^{-6}$ and set $t = 1$. Then compute

$$W_{j,k,l} \quad = \quad e^{-4\alpha\pi^2(\bar{j}^2+\bar{k}^2+\bar{l}^2)t} V_{j,k,l}$$

where $\bar{j}$ is defined as j for $0 \le j < n_1/2$ and $j - n_1$ for $n_1/2 \le j < n_1$. The indices $\bar{k}$ and $\bar{l}$ are similarly defined with n_2 and n_3. Then compute an inverse 3-D DFT on W, using a 3-D FFT routine, and call the result the array X. Finally, compute the complex checksum $\sum_{j=0}^{1023} X_{q,r,s}$ where $q = j \pmod{n_1}$, $r = 3j \pmod{n_2}$ and $s = 5j \pmod{n_3}$. After the checksum for this t has been output, increment t by one. Then repeat the above process, from the computation of W through the incrementing of t, until the step $t = N$ has been completed. In this benchmark, $N = 6$. The V array and the array of exponential terms for $t = 1$ need only be computed once. Note that the array of exponential terms for $t > 1$ can be obtained as the t-th power of the array for $t = 1$.

This completes the definition of the Class A problem. The Class B problem is the same except that $n_1 = 512$, $n_2 = 256$, $n_3 = 256$, and $N = 20$.

Any algorithm may be used for the computation of the 3-D FFTs mentioned above. One algorithm is the following. Assume that the data in the input $n_1 \times n_2 \times n_3$ complex array A are organized so that for each k and l, all elements of the complex vector $(A_{j,k,l},\ 0 \le j < n_1)$ are contained within a single processing node. First perform an n_1-point 1-D FFT on each of these $n_2 n_3$ complex vectors. Then transpose the resulting array into an $n_2 \times n_3 \times n_1$ complex array B. Next, perform an n_2-point 1-D FFT on each of the $n_3 n_1$ first-dimension complex vectors of B. Again note that each of the 1-D FFTs can be performed locally within a single node. Then transpose the resulting array into an $n_3 \times n_1 \times n_2$ complex array C. Finally, perform an n_3-point 1-D FFT on each of the $n_1 n_2$ first-dimension complex vectors of C. Then transpose the resulting array into an $n_1 \times n_2 \times n_3$ complex array D. This array D is the final 3-D FFT result.

Algorithms for performing an individual 1-D complex-to-complex FFT are well known and will not be presented here. Readers are referred to [4, 5, 6, 7, 8] for details. It should be noted that some of these FFTs are "unordered" FFTs, i.e., the results are not in the correct order but instead are scrambled by a bit-reversal permutation. Such FFTs may be employed if desired, but it should be noted that in this case the ordering of the exponential factors in the definition of $W_{j,k,l}$ above must be similarly scrambled in order to obtain the correct results. Also, the final result array X may be scrambled, in which case the checksum calculation will have to be changed accordingly.

It should be noted that individual 1-D FFTs, array transpositions, and even entire 3-D FFT operations may be performed using vendor-supplied library routines. See sections 1.2.2 and 1.2.3 for details.

Operations to be Timed

All of the above operations, including the checksum calculations, must be timed.

Verification Test

The N complex checksums must agree with reference values to within one part in 10^{12}. For the parameter sizes specified above, the reference values are given in table 2.7.

Other Features

- 3-D FFTs are a key part of certain CFD applications, notably large eddy turbulence simulations.
- The 3-D FFT steps require considerable communication for operations such as array transpositions.

2.2.5 Kernel IS: Parallel sort over small integers

L. Dagum

Brief Statement of Problem

Sort N keys in parallel. The keys are generated by the sequential key generation algorithm given below and initially must be uniformly distributed in memory. The initial distribution of the keys can have a great impact on the performance of this benchmark, and the required distribution is discussed in detail below.

	Class A			Class B	
t	Real Part	Imaginary Part	t	Real Part	Imaginary Part
1	504.6735008193	511.4047905510	1	517.7643571579	507.7803458597
2	505.9412319734	509.8809666433	2	515.4521291263	508.8249431599
3	506.9376896287	509.8144042213	3	514.6409228649	509.6208912659
4	507.7892868474	510.1336130759	4	514.2378756213	510.1023387619
5	508.5233095391	510.4914655194	5	513.9626667737	510.3976610617
6	509.1487099959	510.7917842803	6	513.7423460082	510.5948019802
			7	513.5547056878	510.7404165783
			8	513.3910925466	510.8576573661
			9	513.2470705390	510.9577278523
			10	513.1197729984	511.0460304483
			11	513.0070319283	511.1252433800
			12	512.9070537032	511.1968077718
			13	512.8182883502	511.2616233064
			14	512.7393733383	511.3203605551
			15	512.6691062020	511.3735928093
			16	512.6064276004	511.4218460548
			17	512.5504076570	511.4656139760
			18	512.5002331720	511.5053595966
			19	512.4551951846	511.5415130407
			20	512.4146770029	511.5744692211

Table 2.7: FT Benchmark Checksums

Definitions

A sequence of keys, $\{K_i \quad | \quad i = 0, 1, \ldots, N-1\}$, will be said to be *sorted* if it is arranged in non–descending order, i.e., $K_i \leq K_{i+1} \leq K_{i+2} \ldots$. The *rank* of a particular key in a sequence is the index value i that the key would have if the sequence of keys were sorted. *Ranking*, then, is the process of arriving at a rank for all the keys in a sequence. *Sorting* is the process of permuting the the keys in a sequence to produce a sorted sequence. If an initially unsorted sequence, $K_0, K_1, \ldots, K_{N-1}$ has ranks $r(0), r(1), \ldots, r(N-1)$, the sequence becomes sorted when it is rearranged in the order $K_{r(0)}, K_{r(1)}, \ldots, K_{r(N-1)}$. Sorting is said to be *stable* if equal keys retain their original relative order. In other words, a sort is stable only if $r(i) < r(j)$ whenever $K_{r(i)} = K_{r(j)}$ and $i < j$. Stable sorting is not required for this benchmark.

Memory Mapping

The benchmark requires ranking an unsorted sequence of N keys. The initial sequence of keys will be generated in an unambiguous sequential manner as described below. This sequence must be mapped into the memory of the parallel processor in one of the following ways depending on the type of memory system. In all cases, one key will map to one word of memory. Word size must be no less than 32 bits. Once the keys are loaded onto the memory system, they are not to be moved or modified except as required by the procedure described in the Procedure subsection.

Shared Global Memory

All N keys initially must be stored in a contiguous address space. If A_i is used to denote the address of the i^{th} word of memory, then the address space must be $[A_i, A_{i+N-1}]$. The sequence of keys, $K_0, K_1, \ldots, K_{N-1}$, initially must map to this address space as

$$A_{i+j} \longleftarrow MEM(K_j) \quad \text{for} \quad j = 0, 1, \ldots, N-1 \tag{2.1}$$

where $MEM(K_j)$ refers to the address of K_j.

Distributed Memory

In a distributed memory system with p distinct memory units, each memory unit initially must store N_p keys in a contiguous address space, where

$$N_p = N/p \tag{2.2}$$

If A_i is used to denote the address of the i^{th} word in a memory unit, and if P_j is used to denote the j^{th} memory unit, then $P_j \cap A_i$ will denote the address of the i^{th} word in the j^{th} memory unit. Some initial addressing (or "ordering") of memory units must be assumed and adhered to throughout the benchmark. Note that the addressing of the memory units is left completely arbitrary. If N is not evenly divisible by p, then memory units $\{P_j \mid j = 0, 1, \ldots, p-2\}$ will store N_p keys, and memory unit P_{p-1} will store N_{pp} keys, where now

$$\begin{aligned} N_p &= \lfloor N/p + 0.5 \rfloor \\ N_{pp} &= N - (p-1)N_p \end{aligned}$$

In some cases (in particular if p is large) this mapping may result in a poor initial load balance with $N_{pp} >> N_p$. In such cases it may be desirable to use p' memory units to store the keys, where $p' < p$. This is allowed, but the storage of the keys still must follow either equation 2.2 or equations 2.3–2.3 with p' replacing p. In the following we will assume N is evenly divisible by p. The address space in an individual memory unit must be $[A_i, A_{i+N_p-1}]$. If memory units are individually hierarchical, then N_p keys must be stored in a contiguous address space belonging to a single memory hierarchy and A_i then denotes the address of the i^{th} word in that hierarchy. The keys cannot be distributed among different memory hierarchies until after timing begins. The sequence of keys, $K_0, K_1, \ldots, K_{N-1}$, initially must map to this distributed memory as

$$P_k \cap A_{i+j} \longleftarrow MEM(K_{kN_p+j}) \qquad \text{for} \qquad j = 0, 1, \ldots, N_p - 1$$
$$\text{and} \qquad k = 0, 1, \ldots, p-1$$

where $MEM(K_{kN_p+j})$ refers to the address of K_{kN_p+j}. If N is not evenly divisible by p, then the mapping given above must be modified for the case where $k = p - 1$ as

$$P_{p-1} \cap A_{i+j} \longleftarrow MEM(K_{(p-1)N_p+j}) \quad \text{for} \quad j = 0, 1, \ldots, N_{pp} - 1. \quad (2.3)$$

Hierarchical Memory

All N keys initially must be stored in an address space belonging to a single memory hierarchy which will here be referred to as the *main memory.* Note that any memory in the hierarchy which can store all N keys may be used for the initial storage of the keys, and the use of the term "main memory" in the description of this benchmark should not be confused with the more general definition of this term in section 1.2.1. The keys cannot be distributed among different memory hierarchies until after timing begins. The mapping of the keys to the main memory must follow one of either the shared global memory or the distributed memory mappings described above.

The benchmark requires computing the rank of each key in the sequence. The mappings described above define the initial ordering of the keys. For shared global and hierarchical memory systems, the same mapping must be applied to determine the correct ranking. For the case of a distributed memory system, it is permissible for the mapping of keys to memory at the end of the ranking to differ from the initial mapping *only* in the following manner: *the number of keys mapped to a memory unit at the end of the ranking may differ from the initial value, N_p.* It is expected, in a distributed memory machine, that good load balancing of the problem will require changing the initial mapping of the keys and for this reason a different mapping may be used at the end of the ranking. If N_{p_k} is the number of keys in memory unit P_k at the end of the ranking, then the mapping which must be used to determine the correct ranking is given by

$$P_k \cap A_{i+j} \longleftarrow MEM(r(kN_{p_k} + j)) \qquad \text{for} \qquad j = 0, 1, \ldots, N_{p_k} - 1$$
$$\text{and} \qquad k = 0, 1, \ldots, p - 1$$

where $r(kN_{p_k} + j)$ refers to the rank of key $K_{kN_{p_k}+j}$. Note, however, this does not imply that the keys, once loaded into memory, may be moved. Copies of the keys may be made and moved, but the original sequence must remain intact such that each time the ranking process is repeated (Step 4 of Procedure) the original sequence of keys exists (except for the two modifications of Step 4a) and the same algorithm for ranking is applied. Specifically, knowledge obtainable from the communications pattern carried out in the first ranking cannot be used to speed up subsequent rankings and each iteration of Step 4 should be completely independent of the previous iteration.

Key Generation Algorithm

The algorithm for generating the keys makes use of the pseudorandom number generator described in section 2.3. The keys will be in the range

Rank (full)	Full scale	Rank (sample)	Sample code
$r(2112377)$	$104 + i$	$r(48427)$	$0 + i$
$r(662041)$	$17523 + i$	$r(17148)$	$18 + i$
$r(5336171)$	$123928 + i$	$r(23627)$	$346 + i$
$r(3642833)$	$8288932 - i$	$r(62548)$	$64917 - i$
$r(4250760)$	$8388264 - i$	$r(4431)$	$65463 - i$

Table 2.8: Values to be used for partial verification

$[0, B_{max})$. Let r_f be a random fraction uniformly distributed in the range $[0, 1]$, and let K_i be the i^{th} key. The value of K_i is determined as

$$K_i \longleftarrow \lfloor B_{max}(r_{4i+0}+r_{4i+1}+r_{4i+2}+r_{4i+3})/4 \rfloor \quad \text{for} \quad i = 0, 1, \ldots, N-1. \quad (2.4)$$

Note that K_i must be an integer and $\lfloor \cdot \rfloor$ indicates truncation. Four *consecutive* pseudorandom numbers from the pseudorandom number generator must be used for generating each key. All operations before the truncation must be performed in 64-bit double precision. The random number generator must be initialized with $s = 314159265$ as a starting seed.

Partial Verification Test

Partial verification is conducted for each ranking performed. Partial verification consists of comparing a particular subset of ranks with the reference values. The subset of ranks and the reference values are given in table 2.8.

Note that the subset of ranks is selected to be invariant to the ranking algorithm (recall that stability is not required in the benchmark). This is accomplished by selecting for verification only the ranks of unique keys. If a key is unique in the sequence (i.e., there is no other equal key), then it will have a unique rank despite an unstable ranking algorithm. The memory mapping described in the Memory Mapping subsection must be applied.

Full Verification Test

Full verification is conducted after the last ranking is performed. Full verification requires the following:

1. Rearrange the sequence of keys, $\{K_i \mid i = 0, 1, \ldots, N-1\}$, in the

order $\{K_j \quad | \quad j = r(0), r(1), \ldots, r(N-1)\}$, where $r(0), r(1), \ldots, r(N-1)$ is the last computed sequence of ranks.

2. For every K_i from $i = 0 \ldots N-2$ test that $K_i \leq K_{i+1}$.

If the result of this test is true, then the keys are in sorted order. The memory mapping described in the Memory Mapping subsection must be applied.

Procedure

1. In a scalar sequential manner and using the key generation algorithm described above, generate the sequence of N keys.

2. Using the appropriate memory mapping described above, load the N keys into the memory system.

3. Begin timing.

4. Do, for $i = 1$ to I_{max}

   (a) Modify the sequence of keys by making the following two changes:

$$\begin{aligned} K_i &\longleftarrow i \\ K_{i+I_{max}} &\longleftarrow (B_{max} - i) \end{aligned}$$

   (b) Compute the rank of each key.

   (c) Perform the partial verification test described above.

5. End timing.

6. Perform full verification test described above.

Specifications

The specifications given in table 2.9 shall be used in the benchmark. Two sets of values are given, one for Class A and one for Class B.

For partial verification, the reference values given in table 2.8 are to be used. In this table, $r(j)$ refers to the rank of K_j and i is the iteration of Step 4 of the Procedure. Again two sets of values are given, the *Full Scale* set being for the actual benchmark and the *Sample Code* set being for development purposes. It should be emphasized that the benchmark

Parameter	Class A	Class B
N	2^{23}	2^{25}
B_{max}	2^{19}	2^{21}
$seed$	314159265	314159265
I_{max}	10	10

Table 2.9: Parameter values to be used for benchmark

measures the performance based on use of the *Full Scale* values, and the *Sample Code* values are given only as a convenience to the implementor. Also to be supplied to the implementor is Fortran 77 source code for the sequential implementation of the benchmark using the *Sample Code* values and with partial and full verification tests.

2.3 A Pseudorandom Number Generator for the Parallel NAS Kernels

D. Bailey

Suppose that n uniform pseudorandom numbers are to be generated. Set $a = 5^{13}$ and let $x_0 = s$ be a specified initial "seed," i.e., an integer in the range $0 < s < 2^{46}$. Generate the integers x_k for $1 \leq k \leq n$ using the linear congruential recursion

$$x_{k+1} = a x_k \pmod{2^{46}}$$

and return $r_k = 2^{-46} x_k$ as the results. Thus $0 < r_k < 1$, and the r_k are very nearly uniformly distributed on the unit interval. See [2], beginning on page 9 for further discussion of this type of pseudorandom number generator.

Note that any particular value x_k of the sequence can be computed directly from the initial seed s by using the binary algorithm for exponentiation, taking remainders modulo 2^{46} after each multiplication. To be specific, let m be the smallest integer such that $2^m > k$, set $b = s$ and $t = a$. Then repeat the following for i from 1 to m:

$$j \leftarrow k/2$$

$$
\begin{aligned}
b &\leftarrow bt \pmod{2^{46}} \qquad \text{if } \; 2j \neq k \\
t &\leftarrow t^2 \pmod{2^{46}} \\
k &\leftarrow j
\end{aligned}
$$

The final value of b is $x_k = a^k s \pmod{2^{46}}$. See [2] for further discussion of the binary algorithm for exponentiation.

The operation of multiplying two large integers modulo 2^{46} can be implemented using 64 bit floating point arithmetic by splitting the arguments into two words with 23 bits each. To be specific, suppose one wishes to compute $c = ab \pmod{2^{46}}$. Then perform the following steps, where int denotes the greatest integer:

$$
\begin{aligned}
a_1 &\leftarrow \text{int}\,(2^{-23}a) \\
a_2 &\leftarrow a - 2^{23}a_1 \\
b_1 &\leftarrow \text{int}\,(2^{-23}b) \\
b_2 &\leftarrow b - 2^{23}b_1 \\
t_1 &\leftarrow a_1 b_2 + a_2 b_1 \\
t_2 &\leftarrow \text{int}\,(2^{-23}t_1) \\
t_3 &\leftarrow t_1 - 2^{23}t_2 \\
t_4 &\leftarrow 2^{23}t_3 + a_2 b_2 \\
t_5 &\leftarrow \text{int}\,(2^{-46}t_4) \\
c &\leftarrow t_4 - 2^{46}t_5
\end{aligned}
$$

An implementation of the complete pseudorandom number generator algorithm using this scheme produces the same sequence of results on any system that satisfies the following requirements:

- The input multiplier a and the initial seed s, as well as the constants 2^{23}, 2^{-23}, 2^{46} and 2^{-46}, can be represented exactly as 64 bit floating point constants.
- The truncation of a nonnegative 64 bit floating point value less than 2^{24} is exact.
- The addition, subtraction and multiplication of 64 bit floating point values, where the arguments and results are nonnegative whole numbers less than 2^{47}, produce exact results.

- The multiplication of a 64 bit floating point value, which is a nonnegative whole number less than 2^{47}, by the 64 bit floating point value 2^{-m}, $0 \leq m \leq 46$, produces an exact result.

These requirements are met by virtually all scientific computers in use today. Any system based on the IEEE-754 floating point arithmetic standard [1] easily meets these requirements using double precision. However, it should be noted that obtaining an exact power of two constant on some systems requires a loop rather than merely an assignment statement with **.

Other Features

- The period of this pseudorandom number generator is $2^{44} = 1.76 \times 10^{13}$, and it passes all reasonable statistical tests.

- This calculation can be vectorized on vector computers by generating results in batches of size equal to the hardware vector length.

- By using the scheme described above for computing x_k directly, the starting seed of a particular segment of the sequence can be quickly and independently determined. Thus numerous separate segments can be generated on separate processors of a multiprocessor system.

- Once the IEEE-754 floating point arithmetic standard gains universal acceptance among scientific computers, the radix 2^{46} can be safely increased to 2^{52}, although the scheme described above for multiplying two such numbers must be correspondingly changed. This will increase the period of the pseudorandom sequence by a factor of 64 to approximately 1.13×10^{15}.

References

[1] IEEE Standard for Binary Floating Point Numbers, ANSI/IEEE Standard 754-1985. IEEE, New York, 1985.

[2] Knuth, D. E.: The Art of Computer Programming, Vol. 2. Addison-Wesley, 1981.

[3] Duff, I. S.; Erisman, A. M.; and Reid, J. K.: Direct Methods for Sparse Matrices. Clarendon Press, Oxford, 1986.

[4] Agarwal, R. C.; and Cooley, J. W.: Fourier Transform and Convolution Subroutines for the IBM 3090 Vector Facility. IBM Journal of Research and Development, vol. 30, 1986, pp. 145-162.

[5] Bailey, D. H.: A High-Performance FFT Algorithm for Vector Supercomputers. International Journal of Supercomputer Applications, vol. 2, 1988, pp. 82-87.

[6] Pease, M. C.: An Adaptation of the Fast Fourier Transform for Parallel Processing. Journal of the ACM, 1968, pp. 252-264.

[7] Swarztrauber, P. N.: FFT Algorithms for Vector Computers. Parallel Computing, vol. 1, 1984, pp. 45-63.

[8] Swarztrauber, P. N. Multiprocessor FFTs. Parallel Computing, vol. 5, 1987, pp. 197-210.

3 A METHODOLOGY FOR BENCHMARKING SOME CFD KERNELS ON HIGHLY PARALLEL PROCESSORS

Sisira Weeratunga,* Eric Barszcz, Rod Fatoohi,* and V. Venkatakrishnan*

Summary

A collection of iterative PDE solvers embedded in a pseudo application program is proposed for the performance evaluation of CFD codes on highly parallel processors. The pseudo application program is stripped of complexities associated with real CFD application programs, thereby enabling a simpler description of the algorithms. However, it is capable of reproducing the essential computation and data motion characteristics of large scale, state of the art CFD codes. In this chapter, we present a detailed description of the pseudo application program concept. Preceding chapters address our basic approach towards the performance evaluation of parallel supercomputers targeted for use in numerical aerodynamic simulation.

3.1 Introduction

Computational Fluid Dynamics (CFD) is one of the fields in the area of scientific computing that has driven the development of modern vector supercomputers. Availability of these high performance computers has led to impressive advances in the state of the art of CFD, both in terms of the physical complexity of the simulated problems and the development of computational algorithms capable of extracting high levels of sustained performance. However, to carry out the computational simulations essential for future aerospace research, CFD must be able and ready to exploit potential performance and cost/performance gains possible through the use of highly parallel processing technologies. Use of parallel supercomputers appears to be one of the most promising avenues for realizing large complex physical simulations within realistic time and cost constraints. Although many of the current CFD application programs are amenable to a high degree of parallel

*Computer Sciences Corp., Ames Research Center.

computation, performance data on such codes for the current generation of parallel computers often has been less than remarkable. This is especially true for the class of CFD algorithms involving global data dependencies, commonly referred to as the implicit methods. Often the bottleneck is data motion, due to high latencies and inadequate bandwidth.

It is a common practice among computer hardware designers to use the dense linear equation solution subroutine in the LINPACK to represent the scientific computing workload. Unfortunately, the computational structures in most CFD algorithms bear little resemblance to this LINPACK routine, both in terms of its parallelization strategy as well as floating point and memory reference features. Most CFD application codes are characterized by their use of either regular or irregular sparse data structures and associated algorithms. One of the reasons for this situation is the near absence of communication between computer scientists engaged in the design of high performance parallel computers and the computational scientists involved in the development of CFD applications. To be effective, such exchange of information should occur during the early stages of the design process. It appears that one of the contributing factors to this lack of communication is the complexity and confidentiality associated with the state-of-the-art CFD application codes. One way to help the design process is to provide the computer scientists with synthetic CFD application programs, which lack the complexity of a real application, but at the same time retain all the essential computational structures. Such synthetic application codes can be accompanied by detailed and simpler descriptions of the algorithms involved. In return, the performance data on such synthetic application codes can be used to evaluate different parallel supercomputer systems at the procurement stage by the CFD community.

Computational fluid dynamics involves the numerical solution of a system of nonlinear partial differential equations in two or three spatial dimensions, with or without time dependence. The governing partial differential equations, referred to as the Navier-Stokes equations, represent the laws of conservation of mass, momentum and energy applied to a fluid medium in motion. These equations, when supplemented by appropriate boundary and initial conditions, describe a particular physical problem. To obtain a system of equations amenable to solution on a computer requires the discretization of the differential equations through the use of finite difference, finite volume, finite element or spectral methods. The inherent nonlinearities of the governing equations necessitate the use of iterative solution techniques. Over the past years, a variety of efficient numerical algorithms have been developed, all requiring many floating point operations and large amounts of computer memory to achieve a solution with the desired level of accuracy.

In current CFD applications, there are two types of computational meshes used for the spatial discretization process: structured and unstructured. Structured meshes are characterized by a consistent, logical ordering of mesh points, whose connectivity is associated with a rectilinear coordinate system. Computationally, structured meshes give rise to regularly strided memory reference characteristics. In contrast, unstructured meshes offer greater freedom in terms of mesh point distribution, but require the generation and storage of random connectivity information. Computationally, this results in indirect memory addressing with random strides, with its attendant increase in memory bandwidth requirements. The synthetic application codes currently under consideration are restricted to the case of structured meshes.

The numerical solution algorithms used in CFD codes can be broadly categorized as either explicit or implicit, based on the procedure used for the time domain integration. Among the advantages of the explicit schemes are the high degree of easily exploitable parallelism and the localized spatial data dependencies. These properties have resulted in highly efficient implementations of explicit CFD algorithms on a variety of current generation highly parallel processors. However, the explicit schemes suffer from stringent numerical stability bounds and as a result are not optimal for problems that require fine mesh spacing for numerical resolution. In contrast, implicit schemes have less stringent stability bounds and are suitable for problems involving highly stretched meshes. However, their parallel implementation is more difficult and involve local as well as global spatial data dependencies. In addition, some of the implicit algorithms possess limited degrees of exploitable parallelism. At present, we restrict our synthetic applications to three different representative implicit schemes found in a wide spectrum of production CFD codes in use at NASA Ames Research center.

In the remaining sections of this chapter, we describe the development of a collection of synthetic application programs. First we discuss the rationale behind this approach followed by a complete description of three such synthetic applications. We also outline the problem setup along with the associated verification tests, when they are used to benchmark highly parallel systems.

3.2 Rationale

In the past, vector supercomputer performance was evaluated through the use of suites of kernels chosen to characterize generic computational structures present at a site's workload. For example, NAS Kernels [1] were selected to characterize the computational workloads inherent in a majority of algorithms used by the CFD community at Ames Research Center. However, for highly parallel computer systems, this approach is inadequate for the reasons outlined below.

The first stage of the pseudo application development process was the analysis of a variety of implicit CFD codes and the identification of a set of generic computational structures that represented a range of computational tasks embedded in them. As a result, the following computational kernels were selected:

1. Solution of multiple, independent systems of nondiagonally-dominant, block tridiagonal equations with a (5×5) block size.

2. Solution of multiple, independent systems of nondiagonally-dominant, scalar pentadiagonal equations.

3. Regular-sparse, block (5×5) matrix-vector multiplication.

4. Regular-sparse, block (5×5) lower and upper triangular system solution.

These kernels constitute a majority of the computationally-intensive, main building blocks of the CFD programs designed for the numerical solution of three-dimensional (3D), Euler/Navier-Stokes equations using finite-volume/finite-difference discretization on structured grids. Kernels (1) and (2) are representative of the computations associated with the implicit operator in versions of the ARC3D code [2]. These kernels involve global data dependencies. Although they are similar in many respects, there is a fundamental difference with regard to the communication-to-computation ratio. Kernel (3) typifies the computation of the explicit part of almost all CFD algorithms for structured grids. Here all data dependencies are local, with either nearest neighbor or at most next-to-nearest neighbor type dependencies. Kernel (4) represents the computations associated with the implicit operator of a newer class of implicit CFD algorithms, typified by the code INS3D-LU [3]. This kernel may contain only a limited degree of parallelism, relative to the other kernels.

In terms of their parallel implementation, these kernels represent varying characteristics with regard to the following aspects, which are often related:

1. Available degree of parallelism.
2. Level of parallelism and granularity.
3. Data space partitioning strategies.
4. Global vs. local data dependencies.
5. Inter-processor and in-processor data motion requirements.
6. Ratio of communication to computation.

Previous research efforts in adapting algorithms in a variety of flow solvers to the current generation of highly parallel processors have indicated that the overall performance of many CFD codes is critically dependent on the latency and bandwidth of both the in-processor and inter-processor data motion. Therefore, it is important for the integrity of the benchmarking process to faithfully reproduce a majority of the data motions encountered during the execution of applications in which these kernels are embedded. Also, the nature and amount of data motion is dependent on the kernel algorithms along with the associated data structures and the interaction of these kernels among themselves as well as with the remainder of the application that is outside their scope.

To obtain realistic performance data, specification of both the incoming and outgoing data structures of the kernels should mimic those occuring in an application program. The incoming data structure is dependent on the section of the code where the data is generated, not on the kernel. The optimum data structure for the kernel may turn out to be suboptimal for the code segments where the data is generated and vice versa. Similar considerations also apply to the outgoing data structure. Allowing the freedom to choose optimal incoming and outgoing data structures for the kernel as a basis for evaluating its performance is liable to produce results that are not applicable to a complete application code. The overall performance should reflect the cost of data motion that occur between kernels.

In order to reproduce most of the data motions encountered in the execution of these kernels in a typical CFD application, we propose embedding them in a pseudo application code. It is designed for the numerical solution of a synthetic system of nonlinear Partial Differential Equations (PDEs), using iterative techniques similar to those found in CFD applications of interest to NASA Ames Research Center. However, it contains none of the pre- and post-processing required by the full CFD applications, or the interactions of the processors and the I/O subsystem. This can be regarded as a stripped-down version of a CFD application. It retains the basic kernels that are the principal building blocks of the application and admits a majority of the interactions required between these basic routines. Also, the stripped-down version does not represent a fully configured CFD application in terms of system memory requirements. This fact has the potential for creating data partitioning strategies during the parallel implementation of the synthetic problem that may be inappropriate for the full application.

¿From the point of view of functionality, the stripped-down version does not contain the algorithms used to apply boundary conditions as in a real application. It is well known that often the boundary algorithms gives rise to load imbalances and idling of processors in highly parallel systems. Due to the simplification of the boundary algorithms, it is likely that the overall system performance and efficiency data obtained using the stripped-down version may be higher than that of an actual application. This effect is somewhat mitigated by the fact that for most realistic problems, a relatively small time amount of is spent dealing with boundary algorithms when compared to the time spent in dealing with the internal mesh points. Also, most boundary algorithms involve only local data dependencies.

Some of the other advantages of the stripped-down application vs. full application approach are:

1. Allows benchmarking where real application codes are confidential.
2. Easier to manipulate and port from one system to another.
3. Since only the abstract algorithm is specified, facilitates new implementations that are tied closely to the architecture under consideration.
4. Allows easy addition of other existing and emerging CFD algorithms to the benchmarking process.

5. Easily scalable to larger problem sizes.

It should be noted that this synthetic problem differs from a real CFD problem in the following important aspects:

1. In full CFD application codes, a non-orthogonal coordinate transformation [2] is used to map the complex physical domains to the regular computational domains, thereby introducing metric coefficients of the transformation into the governing PDE's and boundary conditions. Such transformations are absent in the synthetic problem, and as a result may have reduced arithmetic complexity and storage requirements.

2. A blend of nonlinear, second- and fourth-difference artificial dissipation terms [4] is used in most of the actual CFD codes, whose coefficients are determined based on the local changes in pressure. In the stripped-down version, only a linear, fourth difference term is used. This reduces the arithmetic and communication complexity needed to compute the added higher-order dissipation terms. However, it should be noted that computation of these artificial dissipation terms involve only local data dependencies, similar to the matrix-vector multiplication kernel.

3. In codes where artificial dissipation is not used, upwind differencing based on either flux-vector splitting [5, 6], flux-difference splitting [7] or Total Variation Diminishing (TVD) schemes [8] is used. The absence of such differencing schemes in the stripped-down version induces effects similar to (2) on the performance data.

4. Absence of turbulence models. Computation of terms representing some turbulence models involve a combination of local and some long-range data dependencies. Arithmetic and communication complexity associated with turbulence models are absent.

In addition, it needs to be emphasized that the stripped-down problem is neither designed nor is suitable for the purposes of evaluating the convergence rates and/or the applicability of various iterative linear system solvers used in computational fluid dynamics applications. As mentioned before, the synthetic problem differs from the real CFD applications in the following important ways:

1. Absence of realistic boundary algorithms.

2. Higher than normal dissipative effects.
3. Lack of upwind differencing effects, based on either flux-vector splitting or TVD schemes.
4. Absence of geometric stiffness introduced through boundary conforming coordinate transformations and highly stretched meshes.
5. Lack of evolution of weak (i.e., C^0-) solutions found in real CFD applications, during the iterative process.
6. Absence of turbulence modelling effects.

Some of these effects tend to suppress the predominantly hyperbolic nature exhibited by the Navier-Stokes equations, when applied to compressible flows at high Reynolds numbers.

3.3 Mathematical Problem Definition

We consider the numerical solution of the following synthetic system of five nonlinear partial differential equations (PDEs):

$$\begin{aligned}
\frac{\partial \mathbf{U}}{\partial \tau} &= \frac{\partial \mathbf{E}(\mathbf{U})}{\partial \xi} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial \eta} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial \zeta} \\
&+ \frac{\partial \mathbf{T}(\mathbf{U}, \mathbf{U}_\xi)}{\partial \xi} + \frac{\partial \mathbf{V}(\mathbf{U}, \mathbf{U}_\eta)}{\partial \eta} + \frac{\partial \mathbf{W}(\mathbf{U}, \mathbf{U}_\zeta)}{\partial \zeta} \\
&+ \mathbf{H}(\mathbf{U}, \mathbf{U}_\xi, \mathbf{U}_\eta, \mathbf{U}_\zeta), \qquad (\tau, \xi, \eta, \zeta) \in D_\tau \times D
\end{aligned} \tag{3.5}$$

with the boundary conditions:

$$\mathcal{B}(\mathbf{U}, \mathbf{U}_\xi, \mathbf{U}_\eta, \mathbf{U}_\zeta) = \mathbf{U}^{\mathbf{B}}(\tau, \xi, \eta, \zeta), \qquad (\tau, \xi, \eta, \zeta) \in D_\tau \times \partial D \tag{3.6}$$

and initial conditions:

$$\mathbf{U} = \mathbf{U}^0(\xi, \eta, \zeta), \qquad (\xi, \eta, \zeta) \in D \quad \text{for} \quad \tau = 0 \tag{3.7}$$

where $D \in \Re^3$ is a bounded domain, ∂D is its boundary and $D_\tau = \{0 \leq \tau \leq T\}$.

Also, the solution to the system of PDEs:

$$\mathbf{U} = \begin{pmatrix} u^{(1)} \\ u^{(2)} \\ u^{(3)} \\ u^{(4)} \\ u^{(5)} \end{pmatrix}$$

defined in $(D \cup \partial D) \times D_\tau$, is a vector function of temporal variable τ and spatial variables (ξ, η, ζ) that form the orthogonal coordinate system in $\Re^3$, i.e.;

$$u^{(m)} = u^{(m)}(\tau, \xi, \eta, \zeta)$$

The vector functions $\mathbf{U}^B$ and $\mathbf{U}^0$ are given and $\mathcal{B}$ is the boundary operator. $\mathbf{E}$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{T}$, $\mathbf{V}$, $\mathbf{W}$ and $\mathbf{H}$ are vector functions with five components each of the form:

$$\mathbf{E} = \begin{pmatrix} e^{(1)} \\ e^{(2)} \\ e^{(3)} \\ e^{(4)} \\ e^{(5)} \end{pmatrix}$$

and $e^{(m)} = e^{(m)}(\mathbf{U})$ etc., are prescribed functions.

The system given by equations 3.5 through 3.7 is in the 'normal-form', i.e., it gives explicitly the time derivatives of all the dependent variables $u^{(1)}, u^{(2)}, \ldots, u^{(5)}$. Consequently, the Cauchy data at $\tau = 0$, given by equation 3.7 permits the calculation of solution $\mathbf{U}(\tau, \xi, \eta, \zeta)$ for $\tau > 0$.

In the current implementation of the synthetic PDE system solver, we seek a steady-state solution of equations 3.5 through 3.7 of the form:

$$\mathbf{U}^* = \mathbf{f}(\xi,\eta,\zeta) = \begin{pmatrix} f^{(1)} \\ f^{(2)} \\ f^{(3)} \\ f^{(4)} \\ f^{(5)} \end{pmatrix}$$

where $f^{(m)} = f^{(m)}(\xi,\eta,\zeta)$ are prescribed functions of the following form:

$$\begin{pmatrix} f^{(1)}(\xi,\eta,\zeta) \\ f^{(2)}(\xi,\eta,\zeta) \\ f^{(3)}(\xi,\eta,\zeta) \\ f^{(4)}(\xi,\eta,\zeta) \\ f^{(5)}(\xi,\eta,\zeta) \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} & \dots & C_{1,13} \\ C_{2,1} & C_{2,2} & \dots & C_{2,13} \\ \vdots & \vdots & \ddots & \vdots \\ C_{5,1} & C_{5,2} & \dots & C_{5,13} \end{pmatrix} \mathbf{e}(\xi,\eta,\zeta)$$

Here, the vector $\mathbf{e}$ is given by

$$\mathbf{e}^T = (1,\ \xi,\ \eta,\ \zeta,\ \xi^2,\ \eta^2,\ \zeta^2,\ \xi^3,\ \eta^3,\ \zeta^3,\ \xi^4,\ \eta^4,\ \zeta^4,)$$

and $C_{m,n}, m = 1,2,\dots,5\ , n = 1,2,\dots,13$ are specified constants. The vector forcing function $\mathbf{H} = [h^{(1)}, h^{(2)}, h^{(3)}, h^{(4)}, h^{(5)}]^T$, where $h^{(m)} = h^{(m)}(\xi,\eta,\zeta)$ is chosen such that the system of PDE's, along with the boundary and initial conditions for $\mathbf{H}$, satisfies the prescribed exact solution, $\mathbf{U}^*$. This implies

$$\begin{aligned} \mathbf{H}^*(\xi,\eta,\zeta) &= -[\frac{\partial \mathbf{E}(\mathbf{U}^*)}{\partial \xi} + \frac{\partial \mathbf{F}(\mathbf{U}^*)}{\partial \eta} + \frac{\partial \mathbf{G}(\mathbf{U}^*)}{\partial \zeta} \\ &\quad + \frac{\partial \mathbf{T}(\mathbf{U}^*, \mathbf{U}^*_\xi)}{\partial \xi} + \frac{\partial \mathbf{V}(\mathbf{U}^*, \mathbf{U}^*_\eta)}{\partial \eta} + \frac{\partial \mathbf{W}(\mathbf{U}^*, \mathbf{U}^*_\zeta)}{\partial \zeta}], \\ &\quad \text{for } (\xi,\eta,\zeta) \in \mathrm{D} \times \mathrm{D}_\tau \end{aligned}$$

The bounded spatial domain D is specified to be the interior of the unit cube $[(0,1) \times (0,1) \times (0,1)]$, i.e.:

$$D = \{(\xi, \eta, \zeta) : 0 < \xi < 1, 0 < \eta < 1, 0 < \zeta < 1\}$$

and its boundary ∂D, is the surface of the unit cube given by

$$\partial D = \{\, (\xi, \eta, \zeta) : \xi = 0 \text{ or } 1\} \cup \{\, (\xi, \eta, \zeta) : \eta = 0 \text{ or } 1\} \cup \{\, (\xi, \eta, \zeta) : \zeta = 0 \text{ or } 1\}$$

The vector functions $\mathbf{E}, \mathbf{F}, \mathbf{G}, \mathbf{T}, \mathbf{V}$ and $\mathbf{W}$ of the synthetic problem are specified to be the following:

$$\mathbf{E} = \begin{pmatrix} -u^{(2)} \\ -[u^{(2)}]^2/u^{(1)} - \phi \\ -[u^{(2)}u^{(3)}]/u^{(1)} \\ -[u^{(2)}u^{(4)}]/u^{(1)} \\ -[u^{(2)}/u^{(1)}][u^{(5)} + \phi] \end{pmatrix}$$

$$\mathbf{F} = \begin{pmatrix} -u^{(3)} \\ -[u^{(2)}u^{(3)}]/u^{(1)} \\ -[u^{(3)}]^2/u^{(1)} - \phi \\ -[u^{(3)}u^{(4)}]/u^{(1)} \\ -[u^{(3)}/u^{(1)}][u^{(5)} + \phi] \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} -u^{(4)} \\ -[u^{(2)}u^{(4)}]/u^{(1)} \\ -[u^{(3)}u^{(4)}]/u^{(1)} \\ -[u^{(4)}]^2/u^{(1)} - \phi \\ -[u^{(4)}/u^{(1)}][u^{(5)} + \phi] \end{pmatrix}$$

where

$$\phi = k_2\{u^{(5)} - 0.5[\frac{([u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2)}{u^{(1)}}]\}$$

Also

$$\mathbf{T} = \begin{pmatrix} d_\xi^{(1)}(\partial u^{(1)}/\partial\xi) \\ d_\xi^{(2)}(\partial u^{(2)}/\partial\xi) + (4/3)k_3k_4(\partial[u^{(2)}/u^{(1)}]/\partial\xi) \\ d_\xi^{(3)}(\partial u^{(3)}/\partial\xi) + k_3k_4(\partial[u^{(3)}/u^{(1)}]/\partial\xi) \\ d_\xi^{(4)}(\partial u^{(4)}/\partial\xi) + k_3k_4(\partial[u^{(4)}/u^{(1)}]/\partial\xi) \\ t^{(5)} \end{pmatrix}$$

where

$$\begin{aligned} t^{(5)} &= d_\xi^{(5)}\frac{\partial u^{(5)}}{\partial\xi} \\ &+0.5(1 - k_1k_5)\frac{\partial}{\partial\xi}(\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}) \\ &+(\frac{1}{6})\frac{\partial}{\partial\xi}[u^{(2)}/u^{(1)}]^2 + k_1k_5\frac{\partial}{\partial\xi}[u^{(5)}/u^{(1)}] \end{aligned}$$

$$\mathbf{V} = \begin{pmatrix} d_\eta^{(1)}(\partial u^{(1)}/\partial\eta) \\ d_\eta^{(2)}(\partial u^{(2)}/\partial\eta) + k_3k_4(\partial[u^{(2)}/u^{(1)}]/\partial\eta) \\ d_\eta^{(3)}(\partial u^{(3)}/\partial\eta) + (4/3)k_3k_4(\partial[u^{(3)}/u^{(1)}]/\partial\eta) \\ d_\eta^{(4)}(\partial u^{(4)}/\partial\eta) + k_3k_4(\partial[u^{(4)}/u^{(1)}]/\partial\eta) \\ v^{(5)} \end{pmatrix}$$

where

$$
\begin{aligned}
v^{(5)} &= d_\eta^{(5)} \frac{\partial u^{(5)}}{\partial \eta} \\
&\quad + 0.5(1 - k_1 k_5) \frac{\partial}{\partial \eta} \left(\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2} \right) \\
&\quad + \left(\frac{1}{6}\right) \frac{\partial}{\partial \eta} [u^{(3)}/u^{(1)}]^2 + k_1 k_5 \frac{\partial}{\partial \eta} [u^{(5)}/u^{(1)}]
\end{aligned}
$$

$$
\mathbf{W} = \begin{pmatrix}
d_\zeta^{(1)}(\partial u^{(1)}/\partial \zeta) \\
d_\zeta^{(2)}(\partial u^{(2)}/\partial \zeta) + k_3 k_4 (\partial [u^{(2)}/u^{(1)}]/\partial \zeta) \\
d_\zeta^{(3)}(\partial u^{(3)}/\partial \zeta) + k_3 k_4 (\partial [u^{(3)}/u^{(1)}]/\partial \zeta) \\
d_\zeta^{(4)}(\partial u^{(4)}/\partial \zeta) + (4./3.) k_3 k_4 (\partial [u^{(4)}/u^{(1)}]/\partial \zeta) \\
w^{(5)}
\end{pmatrix}
$$

where

$$
\begin{aligned}
w^{(5)} &= d_\zeta^{(5)} \frac{\partial u^{(5)}}{\partial \zeta} + 0.5(1 - k_1 k_5) \frac{\partial}{\partial \zeta} \left(\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2} \right) \\
&\quad + \left(\frac{1}{6}\right) \frac{\partial}{\partial \zeta} [u^{(4)}/u^{(1)}]^2 + k_1 k_5 \frac{\partial}{\partial \zeta} [u^{(5)}/u^{(1)}]
\end{aligned}
$$

and k_1, k_2, k_3, k_4, k_5, $d_\xi^{(m)}$, $d_\eta^{(m)}$, $d_\zeta^{(m)}$ $(m = 1, 2, \ldots, 5)$ are given constants.

3.3.1 The boundary conditions

The boundary conditions for the system of PDEs is prescribed to be of the uncoupled Dirichlet type, and is specified to be compatible with $\mathbf{U}^*$, such that

$$
u^{(m)} = f^{(m)}(\xi, \eta, \zeta) \qquad \text{for} \quad (\tau, \xi, \eta, \zeta) \in D_\tau \times \partial D \tag{3.8}
$$

and $m = 1, 2, \ldots, 5$.

3.3.2 The initial conditions

The initial values $\mathbf{U}^0$ in D are set to those obtained by a transfinite, trilinear interpolation [9] of the boundary data given by equation 3.8. Let

$$\begin{aligned} P_\xi^{(m)} &= (1-\xi)\ u^{(m)}(0,\eta,\zeta) + \xi\ u^{(m)}(1,\eta,\zeta) \\ P_\eta^{(m)} &= (1-\eta)\ u^{(m)}(\xi,0,\zeta) + \eta\ u^{(m)}(\xi,1,\zeta) \\ P_\zeta^{(m)} &= (1-\zeta)\ u^{(m)}(\xi,\eta,0) + \zeta\ u^{(m)}(\xi,\eta,1). \end{aligned} \tag{3.9}$$

Then

$$\begin{aligned} u^{(m)}(\tau=0,\xi,\eta,\zeta) &= P_\xi^{(m)} + P_\eta^{(m)} + P_\zeta^{(m)} \\ &\quad -P_\xi^{(m)}P_\eta^{(m)} - P_\eta^{(m)}P_\zeta^{(m)} - P_\zeta^{(m)}P_\xi^{(m)} \\ &\quad +P_\xi^{(m)}P_\eta^{(m)}P_\zeta^{(m)}, \quad \text{for } (\xi,\eta,\zeta) \in D \end{aligned} \tag{3.10}$$

3.4 The Numerical Scheme

Starting from the initial values prescribed by equation 3.10, we seek some discrete approximation $\mathbf{U}_h^* \in D$ to the steady-state solution $\mathbf{U}^*$ of equations 3.5 through 3.7, through the numerical solution of the nonlinear system of PDE's using a pseudo-time marching scheme and a spatial discretization procedure based on finite difference approximations.

3.4.1 Implicit time differencing

The independent temporal variable τ is discretized to produce the set

$$D_\tau = \{\tau_n : n \in [0,N]\}$$

where the discrete time increment $\Delta\tau$ is given by

$$\tau_n = \tau_{n-1} + \Delta\tau = n\Delta\tau$$

Also the discrete approximation of $\mathbf{U}$ on D_τ is denoted by

$$\mathbf{U}(\tau) \approx \mathbf{U}^\tau(n\Delta\tau) = \mathbf{U}^n$$

A generalized single-step temporal differencing scheme for advancing the solution of equations 3.5 through 3.7 is given by [10]

$$\begin{aligned}\Delta\mathbf{U}^n &= \frac{\beta\Delta\tau}{(1+\theta)}\frac{\partial\Delta\mathbf{U}^n}{\partial\tau} + \frac{\Delta\tau}{(1+\theta)}\frac{\partial\mathbf{U}^n}{\partial\tau} + \frac{\theta}{(1+\theta)}\Delta\mathbf{U}^{n-1} \\ &\quad + O[(\beta - \frac{1}{2} - \theta)\Delta\tau^2 + \Delta\tau^3] \qquad (3.11)\end{aligned}$$

where the forward difference operator Δ is defined as

$$\Delta\mathbf{U}^n = \mathbf{U}^{n+1} - \mathbf{U}^n$$

With the appropriate choice for the parameters β and θ, equation 3.11 reproduces many of the well known two- and three-level explicit and implicit schemes. In this particular case, we are interested only in the two-level, first-order accurate, Euler implicit scheme given by $\beta = 1$ and $\theta = 0$, i.e.:

$$\Delta\mathbf{U}^n = \Delta\tau\frac{\partial\Delta\mathbf{U}^n}{\partial\tau} + \Delta\tau\frac{\partial\mathbf{U}^n}{\partial\tau} + O[\Delta\tau^2] \qquad (3.12)$$

Substituting for $(\partial\Delta\mathbf{U}^n/\partial\tau)$ and $(\partial\mathbf{U}/\partial\tau)$ in equation 3.12, using equations 3.5 through 3.7, we get

$$\begin{aligned}\Delta\mathbf{U}^n &= \Delta\tau[\frac{\partial(\Delta\mathbf{E}^n + \Delta\mathbf{T}^n)}{\partial\xi} + \frac{\partial(\Delta\mathbf{F}^n + \Delta\mathbf{V}^n)}{\partial\eta} + \frac{\partial(\Delta\mathbf{G}^n + \Delta\mathbf{W}^n)}{\partial\zeta}] \\ &\quad + \Delta\tau[\frac{\partial(\mathbf{E}+\mathbf{T})^n}{\partial\xi} + \frac{\partial(\mathbf{F}+\mathbf{V})^n}{\partial\eta} + \frac{\partial(\mathbf{G}+\mathbf{W})^n}{\partial\zeta}] + \Delta\tau\mathbf{H}^* \\ &\qquad (3.13)\end{aligned}$$

where $\Delta\mathbf{E}^n = \mathbf{E}^{n+1} - \mathbf{E}^n$ and $\mathbf{E}^{n+1} = \mathbf{E}(\mathbf{U}^{n+1})$ etc.

Equation 3.13 is nonlinear in $\Delta\mathbf{U}^n$ as a consequence of the fact that the increments $\Delta\mathbf{E}^n$, $\Delta\mathbf{F}^n$, $\Delta\mathbf{G}^n$, $\Delta\mathbf{T}^n$, $\Delta\mathbf{V}^n$ and $\Delta\mathbf{W}^n$ are nonlinear functions of the dependent variables $\mathbf{U}$ and its derivatives $\mathbf{U}_\xi$, $\mathbf{U}_\eta$ and $\mathbf{U}_\zeta$. A linear equation with the same temporal accuracy as equation 3.13 can be obtained by a linearization procedure using a local Taylor series expansion in time about $\mathbf{U}^n$ [11, 12];

$$\begin{aligned}\mathbf{E}^{n+1} &= \mathbf{E}^n + (\frac{\partial\mathbf{E}}{\partial\tau})^n\Delta\tau + O(\Delta\tau^2) \\ &= \mathbf{E}^n + (\frac{\partial\mathbf{E}}{\partial\mathbf{U}})^n(\frac{\partial\mathbf{U}}{\partial\tau})^n\Delta\tau + O(\Delta\tau^2) \qquad (3.14)\end{aligned}$$

Also

$$\mathbf{U}^{n+1} = \mathbf{U}^n + (\frac{\partial \mathbf{U}}{\partial \tau})^n \Delta\tau + O(\Delta\tau^2) \tag{3.15}$$

Then, by combining equations 3.14 and 3.15, we get

$$\mathbf{E}^{n+1} = \mathbf{E}^n + (\frac{\partial \mathbf{E}}{\partial \mathbf{U}})^n (\mathbf{U}^{n+1} - \mathbf{U}^n) + O(\Delta\tau^2)$$

or

$$\Delta\mathbf{E}^n = \mathbf{A}^n(\mathbf{U})\Delta\mathbf{U}^n + O(\Delta\tau^2)$$

where $\mathbf{A}(\mathbf{U})$ is the Jacobian matrix $(\partial\mathbf{E}/\partial\mathbf{U})$.

It should be noted that the above non-iterative time-linearization formulation does not lower the formal order of accuracy of temporal discretization in equation 3.13. However, if the steady-state solution is the only objective, the quadratic convergence of the Newton-Raphson method (for sufficiently good initial approximations) is recovered only as $\Delta\tau \to \infty$.

Similarly, the linearization of remaining terms gives

$$\begin{aligned}
\Delta\mathbf{F}^n &= (\frac{\partial \mathbf{F}}{\partial \mathbf{U}})^n \Delta\mathbf{U}^n + O(\Delta\tau^2) \\
&= \mathbf{B}^n \Delta\mathbf{U}^n + O(\Delta\tau^2) \\
\Delta\mathbf{G}^n &= (\frac{\partial \mathbf{G}}{\partial \mathbf{U}})^n \Delta\mathbf{U}^n + O(\Delta\tau^2) \\
&= \mathbf{C}^n \Delta\mathbf{U}^n + O(\Delta\tau^2) \\
\Delta\mathbf{T}^n &= (\frac{\partial \mathbf{T}}{\partial \mathbf{U}})^n \Delta\mathbf{U}^n + (\frac{\partial \mathbf{T}}{\partial \mathbf{U}_\xi})^n \Delta\mathbf{U}_\xi^n + O(\Delta\tau^2) \\
&= \mathbf{M}^n \Delta\mathbf{U}^n + \mathbf{N}^n \Delta\mathbf{U}_\xi^n + O(\Delta\tau^2) \\
&= (\mathbf{M} - \mathbf{N}_\xi)^n \Delta\mathbf{U}^n + \frac{\partial(\mathbf{N}\Delta\mathbf{U})^n}{\partial\xi} + O(\Delta\tau^2) \\
\Delta\mathbf{V}^n &= (\frac{\partial \mathbf{V}}{\partial \mathbf{U}})^n \Delta\mathbf{U}^n + (\frac{\partial \mathbf{V}}{\partial \mathbf{U}_\eta})^n \Delta\mathbf{U}_\eta^n + O(\Delta\tau^2) \\
&= (\mathbf{P} - \mathbf{Q}_\eta)^n \Delta\mathbf{U}^n + \frac{\partial(\mathbf{Q}\Delta\mathbf{U})^n}{\partial\eta} + O(\Delta\tau^2) \\
\Delta\mathbf{W}^n &= (\frac{\partial \mathbf{W}}{\partial \mathbf{U}})^n \Delta\mathbf{U}^n + (\frac{\partial \mathbf{W}}{\partial \mathbf{U}_\zeta})^n \Delta\mathbf{U}_\zeta^n + O(\Delta\tau^2) \\
&= (\mathbf{R} - \mathbf{S}_\zeta)^n \Delta\mathbf{U}^n + \frac{\partial(\mathbf{S}\Delta\mathbf{U})^n}{\partial\zeta} + O(\Delta\tau^2)
\end{aligned} \tag{3.16}$$

where

$$
\begin{aligned}
\mathbf{B}(\mathbf{U}) &= \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \\
\mathbf{C}(\mathbf{U}) &= \frac{\partial \mathbf{G}}{\partial \mathbf{U}} \\
\mathbf{M}(\mathbf{U}, \mathbf{U}_\xi) &= \frac{\partial \mathbf{T}}{\partial \mathbf{U}}; \qquad \mathbf{N}(\mathbf{U}) = \frac{\partial \mathbf{T}}{\partial \mathbf{U}_\xi}; \qquad \mathbf{N}_\xi(\mathbf{U}, \mathbf{U}_\xi) = \frac{\partial \mathbf{N}}{\partial \xi} \\
\mathbf{P}(\mathbf{U}, \mathbf{U}_\eta) &= \frac{\partial \mathbf{V}}{\partial \mathbf{U}}; \qquad \mathbf{Q}(\mathbf{U}) = \frac{\partial \mathbf{V}}{\partial \mathbf{U}_\eta}; \qquad \mathbf{Q}_\eta(\mathbf{U}, \mathbf{U}_\eta) = \frac{\partial \mathbf{Q}}{\partial \eta} \\
\mathbf{R}(\mathbf{U}, \mathbf{U}_\zeta) &= \frac{\partial \mathbf{W}}{\partial \mathbf{U}}; \qquad \mathbf{S}(\mathbf{U}) = \frac{\partial \mathbf{W}}{\partial \mathbf{U}_\zeta}; \qquad \mathbf{S}_\zeta(\mathbf{U}, \mathbf{U}_\zeta) = \frac{\partial \mathbf{S}}{\partial \zeta}
\end{aligned}
$$

When the approximations given by equations 3.16 are introduced into equation 3.13, we obtain the following linear equation for $\Delta \mathbf{U}^n$:

$$
\begin{aligned}
\{\mathbf{I} &- \Delta\tau[\frac{\partial(\mathbf{A} + \mathbf{M} - \mathbf{N}_\xi)^n}{\partial \xi} + \frac{\partial^2(\mathbf{N})^n}{\partial \xi^2} + \frac{\partial(\mathbf{B} + \mathbf{P} - \mathbf{Q}_\eta)^n}{\partial \eta} + \frac{\partial^2(\mathbf{Q})^n}{\partial \eta^2} \\
&+ \frac{\partial(\mathbf{C} + \mathbf{R} - \mathbf{S}_\zeta)^n}{\partial \zeta} + \frac{\partial^2(\mathbf{S})^n}{\partial \zeta^2}]\}\Delta \mathbf{U}^n \\
&= \Delta\tau[\frac{\partial(\mathbf{E} + \mathbf{T})^n}{\partial \xi} + \frac{\partial(\mathbf{F} + \mathbf{V})^n}{\partial \eta} + \frac{\partial(\mathbf{G} + \mathbf{W})^n}{\partial \zeta} + \mathbf{H}^*]
\end{aligned}
\tag{3.17}
$$

It should be noted that the notation of the form

$$
[\frac{\partial(\mathbf{A} + \mathbf{M} - \mathbf{N}_\xi)^n}{\partial \xi}]\Delta \mathbf{U}^n
$$

is used to represent the expressions as such

$$
\frac{\partial[(\mathbf{A} + \mathbf{M} - \mathbf{N}_\xi)^n \Delta \mathbf{U}^n]}{\partial \xi} \qquad \text{etc.}
$$

The left hand side (i.e., **LHS**) of equation 3.17 is referred to as the implicit part and the right hand side (i.e., **RHS**) as the explicit part.

The solution at the advanced time, $\tau = (n+1)\Delta\tau$, is given by

$$
\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta \mathbf{U}^n
$$

The Jacobian matrices for the problem under consideration are given by

$$\mathbf{A} = \begin{pmatrix} 0 & -1 & 0 & 0 & 0 \\ [u^{(2)}/u^{(1)}]^2 - q & (k_2-2)[u^{(2)}/u^{(1)}] & k_2[u^{(3)}/u^{(1)}] & k_2[u^{(4)}/u^{(1)}] & -k_2 \\ [u^{(2)}u^{(3)}]/[u^{(1)}]^2 & -[u^{(3)}/u^{(1)}] & -[u^{(2)}/u^{(1)}] & 0 & 0 \\ [u^{(2)}u^{(4)}]/[u^{(1)}]^2 & -[u^{(4)}/u^{(1)}] & 0 & -[u^{(2)}/u^{(1)}] & 0 \\ a_{51} & a_{52} & k_2[u^{(2)}u^{(3)}]/[u^{(1)}]^2 & k_2[u^{(2)}u^{(4)}]/[u^{(1)}]^2 & -k_1[u^{(2)}/u^{(1)}] \end{pmatrix}$$

where

$$\begin{aligned} q &= (\frac{k_2}{2})\{\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\} \\ a_{51} &= \{k_1[u^{(5)}/u^{(1)}] - 2q\}[\frac{u^{(2)}}{u^{(1)}}] \\ a_{52} &= (\frac{k_2}{2})\{\frac{3[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\} - k_1[\frac{u^{(5)}}{u^{(1)}}] \end{aligned}$$

$$\mathbf{B} = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ [u^{(2)}u^{(3)}]/[u^{(1)}]^2 & -[u^{(3)}/u^{(1)}] & -[u^{(2)}/u^{(1)}] & 0 & 0 \\ [u^{(3)}/u^{(1)}]^2 - q & k_2[u^{(2)}/u^{(1)}] & (k_2-2)[u^{(3)}/u^{(1)}] & k_2[u^{(4)}/u^{(1)}] & -k_2 \\ [u^{(3)}u^{(4)}]/[u^{(1)}]^2 & 0 & -[u^{(4)}/u^{(1)}] & -[u^{(3)}/u^{(1)}] & 0 \\ b_{51} & k_2[u^{(2)}u^{(3)}]/[u^{(1)}]^2 & b_{53} & k_2[u^{(3)}u^{(4)}]/[u^{(1)}]^2 & -k_1[u^{(3)}/u^{(1)}] \end{pmatrix}$$

where

$$\begin{aligned} b_{51} &= \{k_1[u^{(5)}/u^{(1)}] - 2q\}[\frac{u^{(3)}}{u^{(1)}}], \\ b_{53} &= (\frac{k_2}{2})\{\frac{[u^{(2)}]^2 + 3[u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}\} - k_1[\frac{u^{(5)}}{u^{(1)}}] \end{aligned}$$

$$
\mathbf{C} = \begin{pmatrix}
0 & 0 & 0 & -1 & 0 \\
[u^{(2)}u^{(4)}]/[u^{(1)}]^2 & -[u^{(4)}/u^{(1)}] & 0 & -[u^{(2)}/u^{(1)}] & 0 \\
[u^{(3)}u^{(4)}]/[u^{(1)}]^2 & 0 & -[u^{(4)}/u^{(1)}] & -[u^{(3)}/u^{(1)}] & 0 \\
[u^{(4)}/u^{(1)}]^2 - q & k_2[u^{(2)}/u^{(1)}] & k_2[u^{(3)}/u^{(1)}] & (k_2-2)[u^{(4)}/u^{(1)}] & -k_2 \\
c_{51} & k_2[u^{(2)}u^{(4)}]/[u^{(1)}]^2 & k_2[u^{(3)}u^{(4)}]/[u^{(1)}]^2 & c_{54} & -k_1[u^{(4)}/u^{(1)}]
\end{pmatrix}
$$

where

$$
\begin{aligned}
c_{51} &= \{k_1[\frac{u^{(5)}}{u^{(1)}}] - 2q\}[\frac{u^{(4)}}{u^{(1)}}] \\
c_{54} &= (\frac{k_2}{2})\{\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + 3[u^{(4)}]^2}{[u^{(1)}]^2}\} - k_1[\frac{u^{(5)}}{u^{(1)}}]
\end{aligned}
$$

$$
(\mathbf{M} - \mathbf{N}_\xi) = [\emptyset]
$$

$$
\mathbf{N} = \begin{pmatrix}
d_\xi^{(1)} & 0 & 0 & 0 & 0 \\
-[4/3]k_3k_4(u^{(2)}/[u^{(1)}]^2) & \begin{array}{c} d_\xi^{(2)}+ \\ {[4/3]}k_3k_4(1/u^{(1)}) \end{array} & 0 & 0 & 0 \\
-k_3k_4(u^{(3)}/[u^{(1)}]^2) & 0 & \begin{array}{c} d_\xi^{(3)}+ \\ k_3k_4(1/u^{(1)}) \end{array} & 0 & 0 \\
-k_3k_4(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & \begin{array}{c} d_\xi^{(4)}+ \\ k_3k_4(1/u^{(1)}) \end{array} & 0 \\
n_{51} & n_{52} & n_{53} & n_{54} & n_{55}
\end{pmatrix}
$$

where

$$
\begin{aligned}
n_{51} &= -[(4/3)k_3k_4 - k_1k_3k_4k_5]([u^{(2)}]^2/[u^{(1)}]^3) \\
&\quad -[k_3k_4 - k_1k_3k_4k_5]([u^{(3)}]^2/[u^{(1)}]^3) \\
&\quad -[k_3k_4 - k_1k_3k_4k_5]([u^{(4)}]^2/[u^{(1)}]^3) \\
&\quad -k_1k_3k_4k_5(u^{(5)}/[u^{(1)}]^2) \\
n_{52} &= ([4/3]k_3k_4 - k_1k_3k_4k_5)(u^{(2)}/[u^{(1)}]^2) \\
n_{53} &= (k_3k_4 - k_1k_3k_4k_5)(u^{(3)}/[u^{(1)}]^2) \\
n_{54} &= (k_3k_4 - k_1k_3k_4k_5)(u^{(4)}/[u^{(1)}]^2) \\
n_{55} &= d_\xi^{(5)} + (k_1k_3k_4k_5)(1/u^{(1)})
\end{aligned}
$$

$$
(\mathbf{P} - \mathbf{Q}_\eta) = [\emptyset]
$$

$$
\mathbf{Q} = \begin{pmatrix}
d_\eta^{(1)} & 0 & 0 & 0 & 0 \\
-k_3k_4(u^{(2)}/[u^{(1)}]^2) & \begin{array}{c} d_\eta^{(2)}+ \\ k_3k_4(1/u^{(1)}) \end{array} & 0 & 0 & 0 \\
-(4/3)k_3k_4(u^{(3)}/[u^{(1)}]^2) & 0 & \begin{array}{c} d_\eta^{(3)}+ \\ (4/3)k_3k_4(1/u^{(1)}) \end{array} & 0 & 0 \\
-k_3k_4(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & \begin{array}{c} d_\eta^{(4)}+ \\ k_3k_4(1/u^{(1)}) \end{array} & 0 \\
q_{51} & q_{52} & q_{53} & q_{54} & q_{55}
\end{pmatrix}
$$

where

$$
\begin{aligned}
q_{51} &= -[k_3k_4 - k_1k_3k_4k_5]([u^{(2)}]^2/[u^{(1)}]^3) \\
&\quad -[(4/3)k_3k_4 - k_1k_3k_4k_5]([u^{(3)}]^2/[u^{(1)}]^3) \\
&\quad -[k_3k_4 - k_1k_3k_4k_5]([u^{(4)}]^2/[u^{(1)}]^3) \\
&\quad -k_1k_3k_4k_5(u^{(5)}/[u^{(1)}]^2) \\
q_{52} &= (k_3k_4 - k_1k_3k_4k_5)(u^{(2)}/[u^{(1)}]^2) \\
q_{53} &= ([4/3]k_3k_4 - k_1k_3k_4k_5)(u^{(3)}/[u^{(1)}]^2) \\
q_{54} &= (k_3k_4 - k_1k_3k_4k_5)(u^{(4)}/[u^{(1)}]^2) \\
q_{55} &= d_\eta^{(5)} + (k_1k_3k_4k_5)(1/u^{(1)})
\end{aligned}
$$

$$(\mathbf{R} - \mathbf{S}_\zeta) \quad = \quad [\emptyset]$$

$$\mathbf{S} \quad = \quad \begin{pmatrix} d_\zeta^{(1)} & 0 & 0 & 0 & 0 \\ -k_3k_4(u^{(2)}/[u^{(1)}]^2) & \begin{array}{c} d_\zeta^{(2)}+ \\ k_3k_4(1/u^{(1)}) \end{array} & 0 & 0 & 0 \\ -k_3k_4(u^{(3)}/[u^{(1)}]^2) & 0 & \begin{array}{c} d_\zeta^{(3)}+ \\ k_3k_4(1/u^{(1)}) \end{array} & 0 & 0 \\ -(4./3.)k_3k_4(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & \begin{array}{c} d_\zeta^{(4)}+ \\ (4/3)k_3k_4(1/u^{(1)}) \end{array} & 0 \\ s_{51} & s_{52} & s_{53} & s_{54} & s_{55} \end{pmatrix}$$

where

$$\begin{aligned} s_{51} &= -[k_3k_4 - k_1k_3k_4k_5]([u^{(2)}]^2/[u^{(1)}]^3) \\ &\quad -[k_3k_4 - k_1k_3k_4k_5]([u^{(3)}]^2/[u^{(1)}]^3) \\ &\quad -[(4/3)k_3k_4 - k_1k_3k_4k_5]([u^{(4)}]^2/[u^{(1)}]^3) \\ &\quad -k_1k_3k_4k_5(u^{(5)}/[u^{(1)}]^2) \\ s_{52} &= (k_3k_4 - k_1k_3k_4k_5)(u^{(2)}/[u^{(1)}]^2) \\ s_{53} &= (k_3k_4 - k_1k_3k_4k_5)(u^{(3)}/[u^{(1)}]^2) \\ s_{54} &= ([4/3]k_3k_4 - k_1k_3k_4k_5)(u^{(4)}/[u^{(1)}]^2) \\ s_{55} &= d_\zeta^{(5)} + (k_1k_3k_4k_5)(1/u^{(1)}) \end{aligned}$$

3.4.2 Spatial discretization

The independent spatial variables (ξ, η, ζ) are discretized by covering $\bar{D}$, (the closure of D), with a mesh of uniform increments (h_ξ, h_η, h_ζ) in each of the coordinate directions. The mesh points in the region will be identified by the index-triple (i, j, k), where the indices $i \in [1, N_\xi]$, $j \in [1, N_\eta]$ and $k \in [1, N_\zeta]$ correspond to the discretization of ξ, η and ζ coordinates, respectively.

$$D_h \cup \partial D_h \quad = \quad \{(\xi_i, \eta_j, \zeta_k) : 1 \le i \le N_\xi, 1 \le j \le N_\eta, 1 \le k \le N_\zeta\}$$

where

$$\xi_i = (i-1)h_\xi; \qquad \eta_j = (j-1)h_\eta; \qquad \zeta_k = (k-1)h_\zeta \tag{3.18}$$

and the mesh widths are given by

$$h_\xi = 1/(N_\xi - 1); \qquad h_\eta = 1/(N_\eta - 1); \qquad h_\zeta = 1/(N_\zeta - 1)$$

with $(N_\xi, N_\eta, N_\zeta) \in N$ being the number of mesh points in $\xi-, \eta-$ and $\zeta-$directions, respectively.

Then, the set of interior mesh points is given by

$$D_h = \{(\xi_i, \eta_j, \zeta_k) : 2 \leq i \leq (N_\xi - 1), 2 \leq j \leq (N_\eta - 1), 2 \leq k \leq (N_\zeta - 1)\}$$

and the boundary mesh points by

$$\begin{aligned} \partial D_h = \; & \{(\xi_i, \eta_j, \zeta_k) : i \in \{1, N_\xi\}\} \cup \{(\xi_i, \eta_j, \zeta_k) : \\ & j \in \{1, N_\eta\}\} \cup \{(\xi_i, \eta_j, \zeta_k) : k \in \{1, N_\zeta\}\} \end{aligned}$$

Also, the discrete approximation of U in $(\bar{D} \times D_\tau)$ is denoted by

$$U(\tau, \xi, \eta, \zeta) \cong U_h^\tau(n\Delta\tau, (i-1)h_\xi, (j-1)h_\eta, (k-1)h_\zeta) = U_{i,j,k}^n$$

3.4.3 Spatial differencing

The spatial derivatives in equation 3.17 are approximated by the appropriate finite-difference quotients, based on the values of U_h^τ at mesh points in $D_h \cup \partial D_h$. We use three-point, second-order accurate central difference approximations in each of the three coordinate directions.

In the computation of the finite-difference approximation to the **RHS**, the following two general forms of spatial derivatives are encountered, i.e.:

$$\frac{\partial e^{(m)}(\mathbf{U})}{\partial \xi}$$

and

$$\frac{\partial t^{(m)}(\mathbf{U}, \mathbf{U}_\xi)}{\partial \xi}$$

The first form is differenced as (using m-th component vector function $\mathbf{E}$ as an example)

$$\frac{\partial e^{(m)}(\mathbf{U})}{\partial \xi}|_{i,j,k} = (1/2h_\xi)[e^{(m)}(\mathbf{U}_{i+1,j,k}) - e^{(m)}(\mathbf{U}_{i-1,j,k})] + O(h_\xi^2) \tag{3.19}$$

The second form is approximated as (using m-th component of vector function $\mathbf{T}$ as an example)

$$\begin{aligned}\frac{\partial t^{(m)}(\mathbf{U}, \mathbf{U}_\xi)}{\partial \xi}|_{i,j,k} &= (1/h_\xi)\{t^{(m)}[(\frac{\mathbf{U}_{i+1,j,k} + \mathbf{U}_{i,j,k}}{2}), (\frac{\mathbf{U}_{i+1,j,k} - \mathbf{U}_{i,j,k}}{h_\xi})] \\ &- t^{(m)}[(\frac{\mathbf{U}_{i,j,k} + \mathbf{U}_{i-1,j,k}}{2}), (\frac{\mathbf{U}_{i,j,k} - \mathbf{U}_{i-1,j,k}}{h_\xi})]\} + O(h_\xi^2)\end{aligned} \tag{3.20}$$

for $2 \leq i \leq (N_\xi - 1)$. Similar formulas are used in the $\eta-$ and $\zeta-$ directions as well.

During the finite-difference approximation of the spatial derivatives in the implicit part, following three general forms are encountered:

$$\frac{\partial[\mathtt{a}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi}$$

$$\frac{\partial[\mathtt{m}^{(m,l)}(\mathbf{U}, \mathbf{U}_\xi)\Delta u^{(l)}]}{\partial \xi}$$

and

$$\frac{\partial^2[\mathtt{n}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi^2}$$

The first form is approximated by the following:

$$\begin{aligned}\frac{\partial[\mathtt{a}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi}|_{i,j,k} &\approx [(1/2h_\xi)\{\mathtt{a}^{(m,l)}(\mathbf{U}_{i+1,j,k})\}]\Delta u^{(l)}_{i+1,j,k} \\ &- [(1/2h_\xi)\{\mathtt{a}^{(m,l)}(\mathbf{U}_{i-1,j,k})\}]\Delta u^{(l)}_{i-1,j,k}\end{aligned} \tag{3.21}$$

The second form is differenced in the following compact three-point form:

$$
\begin{aligned}
&\frac{\partial[\mathtt{m}^{(m,l)}(\mathbf{U},\mathbf{U}_\xi)\Delta u^{(l)}]}{\partial \xi}|_{i,j,k} \approx \\
&[(1/2h_\xi)\{\mathtt{m}^{(m,l)}[(\frac{\mathbf{U}_{i+1,j,k}+\mathbf{U}_{i,j,k}}{2}),(\frac{\mathbf{U}_{i+1,j,k}-\mathbf{U}_{i,j,k}}{h_\xi})]\}]\Delta u^{(l)}_{i+1,j,k} \\
+\ \ &[(1/2h_\xi)\{\mathtt{m}^{(m,l)}[(\frac{\mathbf{U}_{i+1,j,k}+\mathbf{U}_{i,j,k})}{2}),(\frac{\mathbf{U}_{i+1,j,k}-\mathbf{U}_{i,j,k}}{h_\xi})] \\
-\ \ &\mathtt{m}^{(m,l)}[(\frac{\mathbf{U}_{i,j,k}+\mathbf{U}_{i-1,j,k}}{2}),(\frac{\mathbf{U}_{i,j,k}-\mathbf{U}_{i-1,j,k}}{h_\xi})]\}\Delta u^{(l)}_{i,j,k} \\
-\ \ &[(1/2h_\xi)\{\mathtt{m}^{(m,l)}[(\frac{\mathbf{U}_{i,j,k}+\mathbf{U}_{i-1,j,k}}{2}),(\frac{\mathbf{U}_{i,j,k}-\mathbf{U}_{i-1,j,k}}{h_\xi})]\}]\Delta u^{(l)}_{i-1,j,k}
\end{aligned}
\tag{3.22}
$$

Finally, the third form is differenced as follows:

$$
\begin{aligned}
\frac{\partial^2[\mathtt{n}^{(m,l)}(\mathbf{U})\Delta u^{(l)}]}{\partial \xi^2}|_{i,j,k} \approx\ & [(1/h_\xi^2)\{\mathtt{n}^{(m,l)}(\mathbf{U}_{i+1,j,k})\}]\Delta u^{(l)}_{i+1,j,k} \\
-\ & [(2/h_\xi^2)\{\mathtt{n}^{(m,l)}(\mathbf{U}_{i,j,k})\}]\Delta u^{(l)}_{i,j,k} \\
+\ & [(1/h_\xi^2)\{\mathtt{n}^{(m,l)}(\mathbf{U}_{i-1,j,k})\}]\Delta u^{(l)}_{i-1,j,k}
\end{aligned}
\tag{3.23}
$$

3.4.4 Added higher order dissipation

When central difference type schemes are applied to the solution of the Euler and Navier-Stokes equations, a numerical dissipation model is included, and it plays an important role in determining their success. The form of the dissipation model is quite often a blending of second-difference and fourth-difference dissipation terms [4]. The second-difference dissipation is nonlinear and is used to prevent oscillations in the neighborhood of discontinous (i.e., C^0) solutions, and is negligible where the solution is smooth. The fourth-difference dissipation term is basically linear and is included to suppress high-frequency modes and allow the numerical scheme to converge to a steady state. Near solution discontinuities, it is reduced to zero. It is this term that affects the linear stability of the numerical scheme.

In the current implementation of the synthetic problem, a linear fourth-difference dissipation term of the following form:

$$-\Delta\tau\,\varepsilon\,[h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}+h_\eta^4\frac{\partial^4\mathbf{U}^n}{\partial\eta^4}+h_\zeta^4\frac{\partial^4\mathbf{U}^n}{\partial\zeta^4}]$$

is added to the right hand side of equation 3.5. Here, ε is a specified constant. A similar term with $\mathbf{U}^n$ replaced by $\Delta\mathbf{U}^n$ will appear in the implicit operator, if it is desirable to treat the dissipation term implicitly as well.

In the interior of the computational domain, the fourth-difference term is computed using the following five-point approximation:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k} \approx \mathbf{U}^n_{i+2,j,k}-4\mathbf{U}^n_{i+1,j,k}+6\mathbf{U}^n_{i,j,k}-4\mathbf{U}^n_{i-1,j,k}+\mathbf{U}^n_{i-2,j,k} \tag{3.24}$$

for $4 \le i \le (N_\xi - 3)$.

At the first two interior mesh points belonging to either end of the computational domain, the standard five-point-difference stencil used for the fourth-difference dissipation term is replaced by one-sided or one-sided biased stencils. These modifications are implemented so as to maintain a non-positive definite dissipation matrix for the system of difference equations [16].
Then, at $i = 2$:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k} \approx \mathbf{U}^n_{i+2,j,k}-4\mathbf{U}^n_{i+1,j,k}+5\mathbf{U}^n_{i,j,k}$$

and at $i = 3$:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k} \approx \mathbf{U}^n_{i+2,j,k}-4\mathbf{U}^n_{i+1,j,k}+6\mathbf{U}^n_{i,j,k}-4\mathbf{U}^n_{i-1,j,k}$$

Also at $i = N_\xi - 2$:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k} \approx -4\mathbf{U}^n_{i+1,j,k}+6\mathbf{U}^n_{i,j,k}-4\mathbf{U}^n_{i-1,j,k}+\mathbf{U}^n_{i-2,j,k}$$

and at $i = N_\xi - 1$:

$$h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}|_{i,j,k} \approx 5\mathbf{U}^n_{i,j,k}-4\mathbf{U}^n_{i-1,j,k}+\mathbf{U}^n_{i-2,j,k}$$

Similar difference formulas are also used in the $\eta-$ and $\zeta-$ directions.

Also, for vector functions $\mathbf{T}$, $\mathbf{V}$ and $\mathbf{W}$ used here:

$$(\mathbf{M}-\mathbf{N}_\xi)=0, \qquad (\mathbf{P}-\mathbf{Q}_\eta)=0, \qquad (\mathbf{R}-\mathbf{S}_\zeta)=0$$

Then, for the cases where added higher order dissipation terms are treated only explicitly, equation 3.17 reduces to

$$\begin{aligned}
\{\mathbf{I} \;&-\; \Delta\tau[\frac{\partial(\mathbf{A})^n}{\partial\xi}+\frac{\partial^2(\mathbf{N})^n}{\partial\xi^2}+\frac{\partial(\mathbf{B})^n}{\partial\eta}+\frac{\partial^2(\mathbf{Q})^n}{\partial\eta^2}+\frac{\partial(\mathbf{C})^n}{\partial\zeta}+\frac{\partial^2(\mathbf{S})^n}{\partial\zeta^2}]\}\Delta\mathbf{U}^n \\
&= \Delta\tau[\frac{\partial(\mathbf{E}+\mathbf{T})^n}{\partial\xi}+\frac{\partial(\mathbf{F}+\mathbf{V})^n}{\partial\eta}+\frac{\partial(\mathbf{G}+\mathbf{W})^n}{\partial\zeta}] \\
&- \Delta\tau\,\varepsilon\,[h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}+h_\eta^4\frac{\partial^4\mathbf{U}^n}{\partial\eta^4}+h_\zeta^4\frac{\partial^4\mathbf{U}^n}{\partial\zeta^4}]+\Delta\tau\mathbf{H}^*
\end{aligned} \tag{3.25}$$

When the implicit treatment is extended to the added higher order dissipation terms as well, equation 3.17 takes the following form:

$$\begin{aligned}
\{\mathbf{I} \;&-\; \Delta\tau[\frac{\partial(\mathbf{A})^n}{\partial\xi}+\frac{\partial^2(\mathbf{N})^n}{\partial\xi^2}-\varepsilon h_\xi^4\frac{\partial^4(\mathbf{I})}{\partial\xi^4}+\frac{\partial(\mathbf{B})^n}{\partial\eta}+\frac{\partial^2(\mathbf{Q})^n}{\partial\eta^2}-\varepsilon h_\eta^4\frac{\partial^4(\mathbf{I})}{\partial\eta^4} \\
&+ \frac{\partial(\mathbf{C})^n}{\partial\zeta}+\frac{\partial^2(\mathbf{S})^n}{\partial\zeta^2}-\varepsilon h_\zeta^4\frac{\partial^4(\mathbf{I})}{\partial\zeta^4}]\}\Delta\mathbf{U}^n \\
&= \Delta\tau[\frac{\partial(\mathbf{E}+\mathbf{T})^n}{\partial\xi}+\frac{\partial(\mathbf{F}+\mathbf{V})^n}{\partial\eta}+\frac{\partial(\mathbf{G}+\mathbf{W})^n}{\partial\zeta}] \\
&- \Delta\tau\varepsilon[h_\xi^4\frac{\partial^4\mathbf{U}^n}{\partial\xi^4}+h_\eta^4\frac{\partial^4\mathbf{U}^n}{\partial\eta^4}+h_\zeta^4\frac{\partial^4\mathbf{U}^n}{\partial\zeta^4}]+\Delta\tau\mathbf{H}^*
\end{aligned} \tag{3.26}$$

The modified vector forcing function is given by:

$$\begin{aligned}
\mathbf{H}^*(\xi,\eta,\zeta) \;&=\; -[\frac{\partial\mathbf{E}(\mathbf{U}^*)}{\partial\xi}+\frac{\partial\mathbf{F}(\mathbf{U}^*)}{\partial\eta}+\frac{\partial\mathbf{G}(\mathbf{U}^*)}{\partial\zeta} \\
&+ \frac{\partial\mathbf{T}(\mathbf{U}^*,\mathbf{U}_\xi^*)}{\partial\xi}+\frac{\partial\mathbf{V}(\mathbf{U}^*,\mathbf{U}_\eta^*)}{\partial\eta}+\frac{\partial\mathbf{W}(\mathbf{U}^*,\mathbf{U}_\zeta^*)}{\partial\zeta}] \\
&+ \varepsilon[h_\xi^4\frac{\partial^4\mathbf{U}^*}{\partial\xi^4}+h_\eta^4\frac{\partial^4\mathbf{U}^*}{\partial\eta^4}+h_\zeta^4\frac{\partial^4\mathbf{U}^*}{\partial\zeta^4}], \\
&\quad \text{for } (\xi,\eta,\zeta)\in \mathrm{D}\times\mathrm{D}_\tau
\end{aligned} \tag{3.27}$$

3.4.5 Computation of the explicit part—RHS

The discretized form of the explicit part of equation 3.26 is given by:

$$\begin{aligned}
[\mathbf{RHS}]^n|_{i,j,k} &\approx \Delta\tau[D_\xi(\mathbf{E}+\mathbf{T})^n + D_\eta(\mathbf{F}+\mathbf{V})^n + D_\zeta(\mathbf{G}+\mathbf{W})^n]|_{i,j,k} \\
&- \Delta\tau\,\varepsilon\,[h_\xi^4 D_\xi^4 \mathbf{U}^n + h_\eta^4 D_\eta^4 \mathbf{U}^n + h_\zeta^4 D_\zeta^4 \mathbf{U}^n]|_{i,j,k} \\
&+ \Delta\tau[\mathbf{H}^*]|_{i,j,k}
\end{aligned} \tag{3.28}$$

where D_ξ , D_η and D_ζ are second-order accurate, central spatial differencing operators, defined in D_h. At each point in the mesh, $[\mathbf{RHS}]^n_{i,j,k}$ is a column vector with 5 components. Discretization of added higher order dissipation terms was already discussed in section 4.4. Here we consider the computation of the first term in equation 3.28, using formulas given in equations 3.19 and 3.20:

$$\begin{aligned}
[D_\xi(\mathbf{E}+\mathbf{T})^n &+ D_\eta(\mathbf{F}+\mathbf{V})^n + D_\zeta(\mathbf{G}+\mathbf{W})^n]|_{i,j,k} \\
&= (1/2h_\xi)[\mathbf{E}(\mathbf{U}^n_{i+1,j,k}) - \mathbf{E}(\mathbf{U}^n_{i-1,j,k})] \\
&+ (1/h_\xi)\{\mathbf{T}[(\frac{\mathbf{U}^n_{i+1,j,k}+\mathbf{U}^n_{i,j,k}}{2}),(\frac{\mathbf{U}^n_{i+1,j,k}-\mathbf{U}^n_{i,j,k}}{h_\xi})] \\
&- \mathbf{T}[(\frac{\mathbf{U}^n_{i,j,k}+\mathbf{U}^n_{i-1,j,k}}{2}),(\frac{\mathbf{U}^n_{i,j,k}-\mathbf{U}^n_{i-1,j,k}}{h_\xi})]\} \\
&+ (1/2h_\eta)[\mathbf{F}(\mathbf{U}^n_{i,j+1,k}) - \mathbf{F}(\mathbf{U}^n_{i,j-1,k})] \\
&+ (1/h_\eta)\{\mathbf{V}[(\frac{\mathbf{U}^n_{i,j+1,k}+\mathbf{U}^n_{i,j,k}}{2}),(\frac{\mathbf{U}^n_{i,j+1,k}-\mathbf{U}^n_{i,j,k}}{h_\eta})] \\
&- \mathbf{V}[(\frac{\mathbf{U}^n_{i,j,k}+\mathbf{U}^n_{i,j-1,k}}{2}),(\frac{\mathbf{U}^n_{i,j,k}-\mathbf{U}^n_{i,j-1,k}}{h_\eta})]\} \\
&+ (1/2h_\zeta)[\mathbf{G}(\mathbf{U}^n_{i,j,k+1}) - \mathbf{G}(\mathbf{U}^n_{i,j,k-1})] \\
&+ (1/h_\zeta)\{\mathbf{W}[(\frac{\mathbf{U}^n_{i,j,k+1}+\mathbf{U}^n_{i,j,k}}{2}),(\frac{\mathbf{U}^n_{i,j,k+1}-\mathbf{U}^n_{i,j,k}}{h_\zeta})] \\
&- \mathbf{W}[(\frac{\mathbf{U}^n_{i,j,k}+\mathbf{U}^n_{i,j,k-1}}{2}),(\frac{\mathbf{U}^n_{i,j,k}-\mathbf{U}^n_{i,j,k-1}}{h_\zeta})]\}
\end{aligned} \tag{3.29}$$

for $\{(i,j,k) \in D_h\}$. Also, $[\mathbf{RHS}]^n_{i,j,k} = 0$ for $i = 1$ or N_ξ, $j = 1$ or N_η and $k = 1$ or N_ζ.

This right hand side computation is equivalent in terms of both arithmetic and communication complexity to a regular sparse block (5×5) matrix-vector multiplication, which is the kernel (c). However, its efficient implementation, in this case, does not necessarily require the explicit formation and/or storage of the regular sparse matrix.

3.4.6 Computation of the forcing vector function

Given the analytical form of $\mathbf{U}^*$, the forcing vector function $\mathbf{H}^*$ can simply be evaluated analytically as a function of ξ, η, and ζ, using equation 3.27. This function, along with equation 3.18, can then be used to evaluate $[\mathbf{H}^*]_{i,j,k}$, which is also a colunm vector with 5 components.

Here, we opt for a numerical evaluation of $[\mathbf{H}^*]_{i,j,k}$, using $[\mathbf{U}^*]_{i,j,k}$ and the finite-difference approximations of equations 3.19 and 3.20, as in the case of equation 3.29.

$$
\begin{aligned}
[\mathbf{H}^*]|_{i,j,k} \approx\ & (1/2h_\xi)[\mathbf{E}(\mathbf{U}^*_{i+1,j,k}) - \mathbf{E}(\mathbf{U}^*_{i-1,j,k})] \\
+\ & (1/h_\xi)\{\mathbf{T}[(\frac{\mathbf{U}^*_{i+1,j,k} + \mathbf{U}^*_{i,j,k}}{2}), (\frac{\mathbf{U}^*_{i+1,j,k} - \mathbf{U}^*_{i,j,k}}{h_\xi})] \\
-\ & \mathbf{T}[(\frac{\mathbf{U}^*_{i,j,k} + \mathbf{U}^*_{i-1,j,k}}{2}), (\frac{\mathbf{U}^*_{i,j,k} - \mathbf{U}^*_{i-1,j,k}}{h_\xi})]\} \\
+\ & (1/2h_\eta)[\mathbf{F}(\mathbf{U}^*_{i,j+1,k}) - \mathbf{F}(\mathbf{U}^*_{i,j-1,k})] \\
+\ & (1/h_\eta)\{\mathbf{V}[(\frac{\mathbf{U}^*_{i,j+1,k} + \mathbf{U}^*_{i,j,k}}{2}), (\frac{\mathbf{U}^*_{i,j+1,k} - \mathbf{U}^*_{i,j,k}}{h_\eta})] \\
-\ & \mathbf{V}[(\frac{\mathbf{U}^*_{i,j,k} + \mathbf{U}^*_{i,j-1,k}}{2}), (\frac{\mathbf{U}^*_{i,j,k} - \mathbf{U}^*_{i,j-1,k}}{h_\eta})]\} \\
+\ & (1/2h_\zeta)[\mathbf{G}(\mathbf{U}^*_{i,j,k+1}) - \mathbf{G}(\mathbf{U}^*_{i,j,k-1})] \\
+\ & (1/h_\zeta)\{\mathbf{W}[(\frac{\mathbf{U}^*_{i,j,k+1} + \mathbf{U}^*_{i,j,k}}{2}), (\frac{\mathbf{U}^*_{i,j,k+1} - \mathbf{U}^*_{i,j,k}}{h_\zeta})] \\
-\ & \mathbf{W}[(\frac{\mathbf{U}^*_{i,j,k} + \mathbf{U}^*_{i,j,k-1}}{2}), (\frac{\mathbf{U}^*_{i,j,k} - \mathbf{U}^*_{i,j,k-1}}{h_\zeta})]\}, \\
+\ & \varepsilon\,[h_\xi^4 D_\xi^4 \mathbf{U}^* + h_\eta^4 D_\eta^4 \mathbf{U}^* + h_\zeta^4 D_\zeta^4 \mathbf{U}^*]|_{i,j,k}
\end{aligned}
\tag{3.30}
$$

for $\{(i,j,k) \in D_h\}$. The fourth difference dissipation terms are evaluated using equation 3.24. Also, $[\mathbf{H}]^*_{i,j,k} = 0$ for $i = 1$ or N_ξ, $j = 1$ or N_η and $k = 1$ or N_ζ.

3.4.7 Solution of the system of linear equations

Replacing the spatial derivatives appearing in equation 3.25 by their equivalent difference approximations results in a system of linear equations for $[\Delta \mathbf{U}^n]_{i,j,k}$ for $i \in [2, N_\xi - 1]$, $j \in [2, N_\eta - 1]$ and $k \in [2, N_\zeta - 1]$. Direct solution of this system of linear equations requires a formidable matrix inversion effort, in terms of both the processing time and storage requirements. Therefore, $\Delta \mathbf{U}^n$ is obtained through the use of an iterative method. Here we consider three such iterative methods, involving the kernels (a), (b) and (d). All methods involve some form of approximation to the implicit operator or the **LHS** of equation 3.25. For pseudo-time marching schemes, it is generally sufficient to perform only one iteration per time step.

Approximate factorization (Beam-Warming) algorithm

In this method, the implicit operator in equation 3.25 is approximately factored in the following manner [1, 9]:

$$\{\mathbf{I} - \Delta\tau[\frac{\partial(\mathbf{A})^n}{\partial\xi} + \frac{\partial^2(\mathbf{N})^n}{\partial\xi^2}]\} \times \{I - \Delta\tau[\frac{\partial(\mathbf{B})^n}{\partial\eta} + \frac{\partial^2(\mathbf{Q})^n}{\partial\eta^2}]\}$$
$$\times\{I - \Delta\tau[\frac{\partial(\mathbf{C})^n}{\partial\zeta} + \frac{\partial^2(\mathbf{S})^n}{\partial\zeta^2}]\}\Delta\mathbf{U}^n = \mathbf{RHS}$$

The Beam-Warming algorithm is to be implemented as shown below:

Initialization: Set the boundary values of $\mathbf{U}_{i,j,k}$ for $(i,j,k) \in \partial D_h$ in accordance with equation 3.8. Set the initial values of $\mathbf{U}^0_{i,j,k}$ for $(i,j,k) \in D_h$ in accordance with equation 3.9. Compute the forcing function vector, $\mathbf{H}^*_{i,j,k}$ for $(i,j,k) \in D_h$, using equation 3.30.

Step 1 (Explicit part): Compute the $[\mathbf{RHS}]^n_{i,j,k}$ for $(i,j,k) \in D_h$.
Step 2 (ξ-Sweep of the implicit part): Form and solve the following system of linear equations for $[\Delta\mathbf{U}_1]_{i,j,k}$ for $(i,j,k) \in D_h$:

$$\{\mathbf{I} - \Delta\tau[D_\xi(\mathbf{A})^n + D^2_\xi(\mathbf{N})^n]\}\Delta\mathbf{U}_1 = \mathbf{RHS}$$

Step 3 (η-Sweep of the implicit part): Form and solve the following system of linear equations for $[\Delta\mathbf{U}_2]_{i,j,k}$ for $(i,j,k) \in D_h$:

$$\{\mathbf{I} - \Delta\tau[D_\eta(\mathbf{B})^n + D^2_\eta(\mathbf{Q})^n]\}\Delta\mathbf{U}_2 = \Delta\mathbf{U}_1$$

Step 4 (ζ-Sweep of the implicit part): Form and solve the following system of linear equations for $[\Delta\mathbf{U}^n]_{i,j,k}$ for $(i,j,k) \in D_h$:

$$\{\mathbf{I} - \Delta\tau[D_\zeta(\mathbf{C})^n + D_\zeta^2(\mathbf{S})^n]\}\Delta\mathbf{U}^n = \Delta\mathbf{U}_2$$

Step 5: Update the solution

$$[\mathbf{U}^{n+1}]_{i,j,k} = [\mathbf{U}^n]_{i,j,k} + [\Delta\mathbf{U}^n]_{i,j,k}, \qquad \text{for} \quad (\mathrm{i,j,k}) \in \mathrm{D_h}$$

Steps 1-5 consist of one time-stepping iteration of the Approximate Factorization scheme. The solution of systems of linear equations in each of the steps 2-4 is equivalent to the solution of multiple, independent systems of block tridiagonal equations, with each block being a (5×5) matrix, in the three coordinate directions ξ, η, ζ respectively. For example, the system of equations in the ξ-sweep has the following block tridiagonal structure:

$$\begin{aligned}
[\mathcal{B}_{1,j,k}][\Delta\mathbf{U}_1]_{1,j,k} + [\mathcal{C}_{1,j,k}][\Delta\mathbf{U}_1]_{2,j,k} &= [\mathbf{RHS}]_{1,j,k} \\
[\mathcal{A}_{i,j,k}][\Delta\mathbf{U}_1]_{i-1,j,k} + [\mathcal{B}_{i,j,k}][\Delta\mathbf{U}_1]_{i,j,k} + [\mathcal{C}_{i,j,k}][\Delta\mathbf{U}_1]_{i+1,j,k} &= [\mathbf{RHS}]_{i,j,k} \\
(2 \le i \le N_\xi - 1) & \\
[\mathcal{A}_{N_\xi,j,k}][\Delta\mathbf{U}_1]_{N_\xi-1,j,k} + [\mathcal{B}_{N_\xi,j,k}][\Delta\mathbf{U}_1]_{N_\xi,j,k} &= [\mathbf{RHS}]_{N_\xi,j,k}
\end{aligned}$$

where $(j \in [2, N_\eta - 1])$ and $(k \in [2, N_\zeta - 1])$. Also, $[\mathcal{A}], [\mathcal{B}]$ and $[\mathcal{C}]$ are (5×5) matrices and $[\Delta\mathbf{U}_1]_{i,j,k}$ is a (5×1) column vector.

Here, for $2 \le i \le (N_\xi - 1)$, using equations 3.21 and 3.23:

$$\begin{aligned}
[\mathcal{A}_{i,j,k}] &= -\Delta\tau\{(-1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i-1,j,k})] + (1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i-1,j,k})]\} \\
[\mathcal{B}_{i,j,k}] &= \mathbf{I} + \Delta\tau(2/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i,j,k})] \\
[\mathcal{C}_{i,j,k}] &= -\Delta\tau\{(1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i+1,j,k})] + (1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i+1,j,k})]\}
\end{aligned}$$

Also, $[\mathcal{B}_{1,j,k}] = [\mathbf{I}], [\mathcal{C}_{1,j,k}] = [0], [\mathcal{A}_{N_\xi,j,k}] = [0]$, and $[\mathcal{B}_{N_\xi,j,k}] = [\mathbf{I}]$.

Diagonal form of the approximate factorization algorithm

Here the approximate factorization algorithm of section 4.7.1 is modified so as to transform the coupled systems given by the left hand side of equation 3.26 into an uncoupled diagonal form. This involves further approximations in the treatment of the implicit operator. This diagonalization process targets only the matrices **A**, **B**, and **C** in the implicit operator. Effects of other matrices present in the implicit operator are either ignored or approximated to conform to the resulting diagonal structure.

The diagonalization process is based on the observation that matrices **A**, **B**, and **C** each have a set of real eigenvalues and a complete set of eigenvectors. Therefore, they can be diagonalized through similarity transformations [13]:

$$\begin{aligned}
\mathbf{A} &= \mathbf{T}_\xi \Lambda_\xi \mathbf{T}_\xi^{-1} \\
\mathbf{B} &= \mathbf{T}_\eta \Lambda_\eta \mathbf{T}_\eta^{-1} \\
\mathbf{C} &= \mathbf{T}_\zeta \Lambda_\zeta \mathbf{T}_\zeta^{-1}
\end{aligned} \tag{3.31}$$

with

$$\begin{aligned}
\Lambda_\xi &= \text{diag}\,[-(u^{(2)}/u^{(1)}),\ -(u^{(2)}/u^{(1)}),\ -(u^{(2)}/u^{(1)}),\ -(u^{(2)}/u^{(1)}+a),\ -(u^{(2)}/u^{(1)}-a)] \\
\Lambda_\eta &= \text{diag}\,[-(u^{(3)}/u^{(1)}),\ -(u^{(3)}/u^{(1)}),\ -(u^{(3)}/u^{(1)}),\ -(u^{(3)}/u^{(1)}+a),\ -(u^{(3)}/u^{(1)}-a)] \\
\Lambda_\zeta &= \text{diag}\,[-(u^{(4)}/u^{(1)}),\ -(u^{(4)}/u^{(1)}),\ -(u^{(4)}/u^{(1)}),\ -(u^{(4)}/u^{(1)}+a),\ -(u^{(4)}/u^{(1)}-a)]
\end{aligned}$$

where

$$a = \sqrt{c_1\, c_2 \{ \frac{u^{(5)}}{u^{(1)}} - 0.5\, [\frac{[u^{(2)}]^2 + [u^{(3)}]^2 + [u^{(4)}]^2}{[u^{(1)}]^2}]\}}$$

and $\mathbf{T}_\xi(\mathbf{U})$, $\mathbf{T}_\eta(\mathbf{U})$, and $\mathbf{T}_\zeta(\mathbf{U})$ are the matrices whose columns are the eigenvectors of **A**, **B**, and **C**, respectively. When all other matrices except **A**, **B**, and **C** are ignored, the implicit operator is given by

$$\mathbf{LHS} = [I - \Delta\tau \frac{\partial \mathbf{A}^n}{\partial \xi}][I - \Delta\tau \frac{\partial \mathbf{B}^n}{\partial \eta}][I - \Delta\tau \frac{\partial \mathbf{C}^n}{\partial \zeta}] \tag{3.32}$$

Substituting for $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$, using equation 3.31 in equation 3.32, we get

$$\begin{aligned}
\mathbf{LHS} = &\ [(\mathbf{T}_\xi \mathbf{T}_\xi^{-1})^n - \Delta\tau \frac{\partial (\mathbf{T}_\xi \Lambda_\xi \mathbf{T}_\xi^{-1})^n}{\partial \xi}] \\
&\times [(\mathbf{T}_\eta \mathbf{T}_\eta^{-1})^n - \Delta\tau \frac{\partial (\mathbf{T}_\eta \Lambda_\eta \mathbf{T}_\eta^{-1})^n}{\partial \eta}] \\
&\times [(\mathbf{T}_\zeta \mathbf{T}_\zeta^{-1})^n - \Delta\tau \frac{\partial (\mathbf{T}_\zeta \Lambda_\zeta \mathbf{T}_\zeta^{-1})^n}{\partial \zeta}] \Delta \mathbf{U}^n
\end{aligned} \tag{3.33}$$

A modified form of equation 3.33 is obtained by moving the eigenvector matrices $\mathbf{T}_\xi$, $\mathbf{T}_\eta$, and $\mathbf{T}_\zeta$ outside the spatial differential operators $\partial/\partial\xi$, $\partial/\partial\eta$, and $\partial/\partial\zeta$, respectively [13]:

$$\begin{aligned}\mathbf{LHS} &= \mathbf{T}_\xi^n[\mathbf{I}-\Delta\tau\frac{\partial(\Lambda_\xi)^n}{\partial\xi}](\mathbf{T}_\xi^{-1})^n(\mathbf{T}_\eta)^n[\mathbf{I}-\Delta\tau\frac{\partial(\Lambda_\eta)^n}{\partial\eta}](\mathbf{T}_\eta^{-1})^n(\mathbf{T}_\zeta)^n \\ & \quad [\mathbf{I}-\Delta\tau\frac{\partial(\Lambda_\zeta)^n}{\partial\zeta}](\mathbf{T}_\zeta^{-1})^n\Delta\mathbf{U}^n\end{aligned}$$

This can be written as

$$\mathbf{LHS} = \mathbf{T}_\xi^n[\mathbf{I}-\Delta\tau\frac{\partial(\Lambda_\xi)^n}{\partial\xi}]\bar{\mathbf{N}}[\mathbf{I}-\Delta\tau\frac{\partial(\Lambda_\eta)^n}{\partial\eta}]\bar{\mathbf{P}}[\mathbf{I}-\Delta\tau\frac{\partial(\Lambda_\zeta)^n}{\partial\zeta}](\mathbf{T}_\zeta^{-1})^n\Delta\mathbf{U}^n$$

where

$$\bar{\mathbf{N}} = \mathbf{T}_\xi^{-1}\mathbf{T}_\eta; \qquad \bar{\mathbf{N}}^{-1} = \mathbf{T}_\eta^{-1}\mathbf{T}_\xi$$

$$\bar{\mathbf{P}} = \mathbf{T}_\eta^{-1}\mathbf{T}_\zeta; \qquad \bar{\mathbf{P}}^{-1} = \mathbf{T}_\zeta^{-1}\mathbf{T}_\eta$$

The eigenvector matrices are functions of ξ, η and ζ and therefore this modification introduces further errors of $O(\Delta\tau)$ into the factorization process.

During the factorization process, the presence of the matrices $\mathbf{N}$, $\mathbf{Q}$, and $\mathbf{S}$ in the implicit part were ignored. This is because, in general, the similarity transformations used for diagonalizing $\mathbf{A}$ do not simultaneously diagonalize $\mathbf{N}$. The same is true for the η and ζ factors as well. This necessitates some ad-hoc approximate treatment of the ignored terms, which at the same time preserves the diagonal structure of the implicit operators. Whatever the approach used, additional approximation errors are introduced in the treatment of the implicit operators. We chose to approximate $\mathbf{N}, \mathbf{Q}$, and $\mathbf{S}$ by diagonal matrices in the implicit operators, whose values are given by the spectral radii $\rho(\mathbf{N})$, $\rho(\mathbf{Q})$, and $\rho(\mathbf{S})$, respectively [17]. In addition, we also treat the added fourth-difference dissipation terms implicitly.

$$\begin{aligned}\mathbf{LHS} &= \mathbf{T}_\xi^n[\mathbf{I}-\Delta\tau\{\frac{\partial(\Lambda_\xi)^n}{\partial\xi}+\frac{\partial^2[\rho(\mathbf{N})^n\mathbf{I}]}{\partial\xi^2}-\varepsilon\, h_\xi^4\frac{\partial^4(\mathbf{I})}{\partial\xi^4}\}] \\ & \quad \times\bar{\mathbf{N}}[\mathbf{I}-\Delta\tau\{\frac{\partial(\Lambda_\eta)^n}{\partial\eta}+\frac{\partial^2[\rho(\mathbf{Q})^n\mathbf{I}]}{\partial\eta^2}-\varepsilon\, h_\eta^4\frac{\partial^4(\mathbf{I})}{\partial\eta^4}\}] \\ & \quad \times\bar{\mathbf{P}}[\mathbf{I}-\Delta\tau\{\frac{\partial(\Lambda_\zeta)^n}{\partial\zeta}+\frac{\partial^2[\rho(\mathbf{S})^n\mathbf{I}]}{\partial\zeta^2}-\varepsilon\, h_\zeta^4\frac{\partial^4(\mathbf{I})}{\partial\zeta^4}\}]\mathbf{T}_\zeta^{-1}\Delta\mathbf{U}^n\end{aligned}$$

The matrices $\mathbf{T}_\xi^{-1}$, $\mathbf{T}_\zeta$, $\bar{\mathbf{N}}^{-1}$, and $\bar{\mathbf{P}}^{-1}$ are given by:

$$
\begin{aligned}
\mathbf{T}_\xi^{-1} &= \begin{pmatrix}
(1-[q/a^2]) & c_2[u^{(2)}/u^{(1)}]/a^2 & c_2[u^{(3)}/u^{(1)}]/a^2 & c_2[u^{(4)}/u^{(1)}]/a^2 & -[c_2/a^2] \\
-(u^{(4)}/[u^{(1)}]^2) & 0 & 0 & (1/u^{(1)}) & 0 \\
(u^{(3)}/[u^{(1)}]^2) & 0 & -(1/u^{(1)}) & 0 & 0 \\
\sigma(q-a[u^{(2)}/u^{(1)}]) & \sigma(a-c_2[u^{(2)}/u^{(1)}]) & -\sigma c_2[u^{(3)}/u^{(1)}] & -\sigma c_2[u^{(4)}/u^{(1)}] & \sigma c_2 \\
\sigma(q+a[u^{(2)}/u^{(1)}]) & -\sigma(a+c_2[u^{(2)}/u^{(1)}]) & -\sigma c_2[u^{(3)}/u^{(1)}] & -\sigma c_2[u^{(4)}/u^{(1)}] & \sigma c_2
\end{pmatrix} \\
\mathbf{T}_\zeta &= \begin{pmatrix}
0 & 0 & 1 & \alpha & \alpha \\
0 & -u^{(1)} & [u(2)/u(1)] & \alpha[u(2)/u(1)] & \alpha[u(2)/u(1)] \\
u^{(1)} & 0 & [u(3)/u(1)] & \alpha[u(3)/u(1)] & \alpha[u(3)/u(1)] \\
0 & 0 & [u(4)/u(1)] & \alpha([u(4)/u(1)]+a) & \alpha([u(4)/u(1)]-a) \\
u^{(3)} & -u^{(2)} & [q/c_2] & \alpha[(q+a^2)/c_2+a(u^{(4)}/u^{(1)})] & \alpha[(q+a^2)/c_2-a(u^{(4)}/u^{(1)})]
\end{pmatrix} \\
\bar{\mathbf{N}}^{-1} &= \begin{pmatrix}
0 & -1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \\
0 & 0 & -1/\sqrt{2} & 1/2 & 1/2 \\
0 & 0 & 1/\sqrt{2} & 1/2 & 1/2
\end{pmatrix} \\
\bar{\mathbf{P}}^{-1} &= \begin{pmatrix}
0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \\
0 & 0 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
-1/\sqrt{2} & 0 & 0 & 1/2 & 1/2 \\
1/\sqrt{2} & 0 & 0 & 1/2 & 1/2
\end{pmatrix}
\end{aligned}
$$

where $\sigma = 1/(\sqrt{2}\,u^{(1)}a)$ and $\alpha = [u^{(1)}/(\sqrt{2}\,a)]$.

In addition, the spectral radii of the matrices $\mathbf{N}, \mathbf{Q}$, and $\mathbf{S}$ are given by:

$$
\rho(\mathbf{N}) = \max(\begin{array}{c}
d_\xi^{(1)} \\
d_\xi^{(2)} + [4/3]k_3k_4[1/u^{(1)}] \\
d_\xi^{(3)} + k_3k_4[1/u^{(1)}] \\
d_\xi^{(4)} + k_3k_4[1/u^{(1)}] \\
d_\xi^{(5)} + k_1k_3k_4k_5[1/u^{(1)}]\,)
\end{array}
$$

$$\rho(\mathbf{Q}) = \max(\begin{array}{c} d_\eta^{(1)} \\ d_\eta^{(2)} + k_3k_4[1/u^{(1)}] \\ d_\eta^{(3)} + [4/3]k_3k_4[1/u^{(1)}] \\ d_\eta^{(4)} + k_3k_4[1/u^{(1)}] \\ d_\eta^{(5)} + k_1k_3k_4k_5[1/u^{(1)}] \end{array})$$

$$\rho(\mathbf{S}) = \max(\begin{array}{c} d_\zeta^{(1)} \\ d_\zeta^{(2)} + k_3k_4[1/u^{(1)}] \\ d_\zeta^{(3)} + k_3k_4[1/u^{(1)}] \\ d_\zeta^{(4)} + [4/3]k_3k_4[1/u^{(1)}] \\ d_\zeta^{(5)} + k_1k_3k_4k_5[1/u^{(1)}] \end{array})$$

The explicit part of the diagonal algorithm is exactly the same as in equation 3.27 and all the approximations are restricted to the implicit operator. Therefore, if the diagonal algorithm converges, the steady-state solution will be identical to the one obtained without diagonalization. However, the convergence behavior of the diagonalized algorithm would be different.

The Diagonalized Approximate Factorization algorithm is to be implemented in the following order:

Initialization: Set the boundary values of $\mathbf{U}_{i,j,k}$ for $(i,j,k) \in \partial D_h$ in accordance with equation 3.8. Set the initial values of $\mathbf{U}^0_{i,j,k}$ for $(i,j,k) \in D_h$ in accordance withequation 3.9. Compute the forcing function vector, $\mathbf{H}^*_{i,j,k}$ for $(i,j,k) \in D_h$, using equation 3.30.
Step 1 (Explicit part): Compute $[\mathbf{RHS}]^n_{i,j,k}$ for $(i,j,k) \in D_h$.
Step 2: Perform the matrix-vector multiplication

$$[\Delta\mathbf{U}_1] = (\mathbf{T}_\xi^{-1})^n[\mathbf{RHS}]$$

Step 3 (ξ-Sweep of the Implicit part): Form and solve the following system of linear equations for $\Delta\mathbf{U}_2$:

$$\{I - \Delta\tau[D_\xi(\Lambda_\xi)^n] - \Delta\tau[D_\xi^2(\rho(\mathbf{N})^n\mathbf{I})] + \Delta\tau[\varepsilon\, h_\xi^4 D_\xi^4(\mathbf{I})]\}[\Delta\mathbf{U}_2] = [\Delta\mathbf{U}_1]$$

Step 4: Perform the matrix-vector multiplication

$[\Delta\mathbf{U}_3] = \bar{\mathbf{N}}^{-1}[\Delta\mathbf{U}_2]$

Step 5 (η-Sweep of the implicit part): Form and solve the following system of linear equations for $\Delta\mathbf{U}_4$:

$\{I - \Delta\tau[D_\eta(\Lambda_\eta)^n] - \Delta\tau[D_\eta^2(\rho(\mathbf{Q})^n\mathbf{I})] + \Delta\tau[\varepsilon\, h_\eta^4 D_\eta^4(\mathbf{I})]\}[\Delta\mathbf{U}_4] = [\Delta\mathbf{U}_3]$

Step 6: Perform the matrix-vector multiplication

$[\Delta\mathbf{U}_5] = \bar{\mathbf{P}}^{-1}[\Delta\mathbf{U}_4]$

Step 7 (ζ-Sweep of the implicit part): Form and solve the following system of linear equations for $\Delta\mathbf{U}_6$:

$\{I - \Delta\tau[D_\zeta(\Lambda_\zeta)^n] - \Delta\tau[D_\zeta^2(\rho(\mathbf{S})^n\mathbf{I})] + \Delta\tau[\varepsilon\, h_\zeta^4 D_\zeta^4(\mathbf{I})]\}[\Delta\mathbf{U}_6] = [\Delta\mathbf{U}_5]$

Step 8: Perform the matrix-vector multiplication

$[\Delta\mathbf{U}^n] = \mathbf{T}_\zeta[\Delta\mathbf{U}_6]$

Step 9: Update the solution

$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta\mathbf{U}^n$

Steps 1-9 constitute of one iteration of the Diagonal Form of the Approximate Factorization algorithm. The new implicit operators are block pentadiagonal. However, the blocks are diagonal in form, so that the operators simplify into five independent scalar pentadiagonal systems. Therefore, each of the steps 3, 5, and 7 involve the solution of multiple, independent systems of scalar pentadiagonal equations in the three coordinate directions ξ, η and ζ respectively. For example, each of the five independent scalar pentadiagonal systems in the ξ-sweep has the following structure:

$$\begin{aligned}
\mathtt{c}_{1,j,k}[\Delta\mathbf{U}_2]^{(m)}_{1,j,k} + \mathtt{d}_{1,j,k}[\Delta\mathbf{U}_2]^{(m)}_{2,j,k} + \mathtt{e}_{1,j,k}[\Delta\mathbf{U}_2]^{(m)}_{3,j,k} &= [\Delta\mathbf{U}_1]^{(m)}_{1,j,k} \\
\mathtt{b}_{2,j,k}[\Delta\mathbf{U}_2]^{(m)}_{1,j,k} + \mathtt{c}_{2,j,k}[\Delta\mathbf{U}_2]^{(m)}_{2,j,k} + \mathtt{d}_{2,j,k}[\Delta\mathbf{U}_2]^{(m)}_{3,j,k} + \mathtt{e}_{2,j,k}[\Delta\mathbf{U}_2]^{(m)}_{4,j,k} &= [\Delta\mathbf{U}_1]^{(m)}_{2,j,k} \\
\mathtt{a}_{i,j,k}[\Delta\mathbf{U}_2]^{(m)}_{i-2,j,k} + \mathtt{b}_{i,j,k}[\Delta\mathbf{U}_2]^{(m)}_{i-1,j,k} + \mathtt{c}_{i,j,k}[\Delta\mathbf{U}_2]^{(m)}_{i,j,k} + \mathtt{d}_{i,j,k}[\Delta\mathbf{U}_2]^{(m)}_{i+1,j,k} & \\
+\mathtt{e}_{i,j,k}[\Delta\mathbf{U}_2]^{(m)}_{i+2,j,k} &= [\Delta\mathbf{U}_1]^{(m)}_{i,j,k}, \\
\text{for } 3 \le \mathrm{i} \le (\mathrm{N}_\xi - 2) & \\
\mathtt{a}_{N_\xi-1,j,k}[\Delta\mathbf{U}_2]^{(m)}_{N_\xi-3,j,k} + \mathtt{b}_{N_\xi-1,j,k}[\Delta\mathbf{U}_2]^{(m)}_{N_\xi-2,j,k} + \mathtt{c}_{N_\xi-1,j,k}[\Delta\mathbf{U}_2]^{(m)}_{N_\xi-1,j,k} & \\
+\mathtt{d}_{N_\xi-1,j,k}[\Delta\mathbf{U}_2]^{(m)}_{N_\xi,j,k} &= [\Delta\mathbf{U}_1]^{(m)}_{N_\xi-1,j,k} \\
\mathtt{a}_{N_\xi,j,k}[\Delta\mathbf{U}_2]^{(m)}_{N_\xi-2,j,k} + \mathtt{b}_{N_\xi,j,k}[\Delta\mathbf{U}_2]^{(m)}_{N_\xi-1,j,k} + \mathtt{c}_{N_\xi,j,k}[\Delta\mathbf{U}_2]^{(m)}_{N_\xi,j,k} &= [\Delta\mathbf{U}_1]^{(m)}_{N_\xi,j,k}
\end{aligned}$$

where, ($j \in [2, N_\eta - 1]$), ($k \in [2, N_\zeta - 1]$) and ($m \in [1, 5]$). The elements of the pentadiagonal matrix are given by the following:

$$
\begin{aligned}
\mathtt{c}_{1,j,k} &= 1; \qquad \mathtt{d}_{1,j,k} = 0; \qquad \mathtt{e}_{1,j,k} = 0 \\
\mathtt{b}_{2,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{1,j,k} \\
\mathtt{c}_{2,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{2,j,k} + \Delta\tau\,\varepsilon\,(5) \\
\mathtt{d}_{2,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{3,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{3,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{e}_{2,j,k} &= \Delta\tau\,\varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{a}_{3,j,k} &= 0.0, \\
\mathtt{b}_{3,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{2,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{2,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{c}_{3,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{3,j,k} + \Delta\tau\,\varepsilon\,(6) \\
\mathtt{d}_{3,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{4,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{4,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{e}_{3,j,k} &= \Delta\tau\,\varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{a}_{i,j,k} &= \Delta\tau\,\varepsilon \\
\mathtt{b}_{i,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{i-1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{i-1,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{c}_{i,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{i,j,k} + \Delta\tau\,\varepsilon\,(6) \\
\mathtt{d}_{i,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{i+1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{i+1,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{e}_{i,j,k} &= \Delta\tau\,\varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{a}_{N_\xi-2,j,k} &= \Delta\tau\,\varepsilon \\
\mathtt{b}_{N_\xi-2,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\xi-3,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\xi-3,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{c}_{N_\xi-2,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\xi-2,j,k} + \Delta\tau\,\varepsilon\,(6) \\
\mathtt{d}_{N_\xi-2,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\xi-1,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\xi-1,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{e}_{N_\xi-2,j,k} &= 0.0.
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{a}_{N_\xi-1,j,k} &= \Delta\tau\,\varepsilon \\
\mathtt{b}_{N_\xi-1,j,k} &= \Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\xi-2,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\xi-2,j,k} + \Delta\tau\,\varepsilon\,(-4) \\
\mathtt{c}_{N_\xi-1,j,k} &= 1. + \Delta\tau(2/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\xi-1,j,k} + \Delta\tau\,\varepsilon\,(5) \\
\mathtt{d}_{N_\xi-1,j,k} &= -\Delta\tau(1/2h_\xi)[(\Lambda_\xi)^n]^{(m,m)}_{N_\xi,j,k} - \Delta\tau(1/h_\xi^2)[\rho(\mathbf{N})^n]_{N_\xi,j,k}
\end{aligned}
$$

$$\mathtt{a}_{N_\xi,j,k} = 0.; \qquad \mathtt{b}_{N_\xi,j,k} = 0.; \qquad \mathtt{c}_{N_\xi,j,k} = 1$$

Symmetric successive over-relaxation algorithm

In this method, the system of linear equations obtained after replacing the spatial derivatives appearing in the implicit operator in equation 3.25 by second-order accurate, central finite difference operators, is solved using the symmetric, successive over-relaxation scheme. Let the linear system be given by

$$[\mathbf{K}^n][\Delta\mathbf{U}^n] = [\mathbf{RHS}^n]$$

where

$$\mathbf{K}^n = \{\mathbf{I} - \Delta\tau[D_\xi\mathbf{A}^n + D_\xi^2\mathbf{N}^n + D_\eta\mathbf{B}^n + D_\eta^2\mathbf{Q}^n + D_\zeta\mathbf{C}^n + D_\zeta^2\mathbf{S}^n]\}\Delta\mathbf{U}^n, \quad \text{for } (i,j,k) \in D_h$$

It is specified that the unknowns be ordered corresponding to the gridpoints, lexicographically, such that the index in ξ-direction runs fastest, followed by the index in η-direction and finally in ζ-direction. The finite-difference discretization matrix $\mathbf{K}$, resulting from such an ordering has a very regular, banded structure. There are altogether seven block diagonals, each with a (5×5) block size.

The matrix $\mathbf{K}$ can be written as the sum of the matrices $\mathbf{D}$, $\mathbf{Y}$ and $\mathbf{Z}$:

$$\mathbf{K}^n = \mathbf{D}^n + \mathbf{Y}^n + \mathbf{Z}^n$$

where

$$\begin{aligned}
\mathbf{D}^n &= \text{Main block} - \text{diagonal of } \mathbf{K}^n \\
\mathbf{Y}^n &= \text{Three sub} - \text{block} - \text{diagonals of } \mathbf{K}^n \\
\mathbf{Z}^n &= \text{Three super} - \text{block} - \text{diagonals of } \mathbf{K}^n
\end{aligned}$$

Therefore, $\mathbf{D}$ is a block-diagonal matrix, while $\mathbf{Y}$ and $\mathbf{Z}$ are strictly lower and upper triangular, respectively. Then the point-SSOR iteration scheme can be written as [14, 15]:

$$[\mathbf{X}^n][\Delta\mathbf{U}^n] = [\mathbf{RHS}]$$

where

$$\begin{aligned}\mathbf{X}^n &= \omega(2.-\omega)(\mathbf{D}^n+\omega\mathbf{Y}^n)(\mathbf{D}^n)^{-1}(\mathbf{D}^n+\omega\mathbf{Z}^n)\\ &= \omega(2.-\omega)(\mathbf{D}^n+\omega\mathbf{Y}^n)(\mathbf{I}+\omega(\mathbf{D}^n)^{-1}\mathbf{Z}^n)\end{aligned}$$

and $\omega \in (0.,2.)$ is the over-relaxation factor (a specified constant). The SSOR algorithm is to be implemented in following manner:

Initialization: Set the boundary values of $\mathbf{U}_{i,j,k}$ in accordance with equation 3.8. Set the initial values of $\mathbf{U}^0_{i,j,k}$ in accordance with equation 3.9. Compute the forcing function vector, $\mathbf{H}^*_{i,j,k}$. for $(i,j,k) \in D_h$, using equation 3.30.

Step 1 (The explicit part): Compute $[\mathbf{RHS}]^n_{i,j,k}$ for $(i,j,k) \in D_h$.
Step 2 (Lower triangular system solution): Form and solve the following regular, sparse, block lower triangular system to get $[\Delta\mathbf{U}_1]$:

$$(\mathbf{D}^n+\omega\mathbf{Y}^n)[\Delta\mathbf{U}_1] = [\mathbf{RHS}]$$

Step 3 (Upper triangular system solution): Form and solve the following regular, sparse, block upper triangular system to get $[\Delta\mathbf{U}^n]$:

$$(\mathbf{I}+\omega(\mathbf{D}^n)^{-1}\mathbf{Z}^n)[\Delta\mathbf{U}^n] = [\Delta\mathbf{U}_1]$$

Step 4: Update the solution:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + [1/\omega(2.-\omega)]\Delta\mathbf{U}^n$$

Steps 1-4 constitute one complete iteration of the SSOR scheme. The l-th block row of the matrix $\mathbf{K}$ has the following structure:

$$\begin{aligned}&[\mathcal{A}_l][\Delta\mathbf{U}^n]_{l-(N_\xi-2)(N_\eta-2)} + [\mathcal{B}_l][\Delta\mathbf{U}^n]_{l-(N_\xi-2)} + [\mathcal{C}_l][\Delta\mathbf{U}^n]_{l-1} + [\mathcal{D}_l][\Delta\mathbf{U}^n]_l\\ &\quad + [\mathcal{E}_l][\Delta\mathbf{U}^n]_{l+1} + [\mathcal{F}_l][\Delta\mathbf{U}^n]_{l+(N_\xi-2)} + [\mathcal{G}_l][\Delta\mathbf{U}^n]_{l+(N_\xi-2)(N_\eta-2)} = [\mathbf{RHS}]_l\end{aligned}$$

where $l = i+(N_\xi-2)(j-1)+(N_\xi-2)(N_\eta-2)(k-1)$ and $(i,j,k)\in D_h$. The (5×5) matrices are given by:

$$\begin{aligned}[\mathcal{A}_l] &= -\Delta\tau(-1/2h_\zeta)[\mathbf{C}(\mathbf{U}^n_{i,j,k-1})] - \Delta\tau(1/h_\zeta^2)[\mathbf{S}(\mathbf{U}^n_{i,j,k-1})]\\ [\mathcal{B}_l] &= -\Delta\tau(-1/2h_\eta)[\mathbf{B}(\mathbf{U}^n_{i,j-1,k})] - \Delta\tau(1/h_\eta^2)[\mathbf{Q}(\mathbf{U}^n_{i,j-1,k})]\\ [\mathcal{C}_l] &= -\Delta\tau(-1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i-1,j,k})] - \Delta\tau(1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i-1,j,k})]\\ [\mathcal{D}_l] &= \mathbf{I} + \Delta\tau(2/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i,j,k})] + \Delta\tau(2/h_\eta^2)[\mathbf{Q}(\mathbf{U}^n_{i,j,k})] + \Delta\tau(2/h_\zeta^2)[\mathbf{S}(\mathbf{U}^n_{i,j,k})]\\ [\mathcal{E}_l] &= -\Delta\tau(1/2h_\xi)[\mathbf{A}(\mathbf{U}^n_{i+1,j,k})] - \Delta\tau(1/h_\xi^2)[\mathbf{N}(\mathbf{U}^n_{i+1,j,k})]\\ [\mathcal{F}_l] &= -\Delta\tau(1/2h_\eta)[\mathbf{B}(\mathbf{U}^n_{i,j+1,k})] - \Delta\tau(1/h_\eta^2)[\mathbf{Q}(\mathbf{U}^n_{i,j+1,k})]\\ [\mathcal{G}_l] &= -\Delta\tau(1/2h_\zeta)[\mathbf{C}(\mathbf{U}^n_{i,j,k+1})] - \Delta\tau(1/h_\zeta^2)[\mathbf{S}(\mathbf{U}^n_{i,j,k+1})]\end{aligned}$$

Also, $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are the elements in the l-th block row of the matrix $\mathbf{Y}$, whereas $\mathcal{E}$,$\mathcal{F}$ and $\mathcal{G}$ are the elements in the l-the block row of the matrix $\mathbf{Z}$.

3.5 Benchmarks

There are three benchmarks associated with the three numerical schemes described in section 4, for the solution of system of linear equations given by equation 3.25. Each benchmark consists of running one of the three numerical schemes for N_s time steps with given values for N_ξ, N_η, N_ζ and $\Delta\tau$.

3.5.1 Verification test

For each of the benchmarks, at the completion of N_s time steps, compute the following:

1. Root Mean Square norms $RMSR(m)$, of the residual vectors $[\mathbf{RHS}^{(m)}]^n_{i,j,k}$, for $m = 1, 2, \ldots, 5$ where $n = N_s$, and $(i, j, k) \in D_h$

$$RMSR(m) = \sqrt{\frac{\sum_{k=2}^{(N_\zeta-1)} \sum_{j=2}^{(N_\eta-1)} \sum_{i=2}^{(N_\xi-1)} \{[\mathbf{RHS}^{(m)}]^{N_s}_{(i,j,k)}\}^2}{(N_\xi - 2)(N_\eta - 2)(N_\zeta - 2)}}$$

2. Root Mean Square norms $RMSE(m)$ of the error vectors $\{[\mathbf{U}^*]^{(m)}_{i,j,k} - [\mathbf{U}^n]^{(m)}_{i,j,k}\}$, for $m = 1, 2, \ldots, 5$ where $n = N_s$, and $(i, j, k) \in D_h$

$$RMSE(m) = \sqrt{\frac{\sum_{k=2}^{(N_\zeta-1)} \sum_{j=2}^{(N_\eta-1)} \sum_{i=2}^{(N_\xi-1)} \{[\mathbf{U}^*]^{(m)}_{(i,j,k)} - [\mathbf{U}^{N_s}]^{(m)}_{(i,j,k)}\}^2}{(N_\xi - 2)(N_\eta - 2)(N_\zeta - 2)}}$$

3. The numerically evaluated surface integral given by

$$\begin{aligned} I = 0.25\{ \quad & \textstyle\sum_{j=j_1}^{j_2-1} \sum_{i=i_1}^{i_2-1} h_\xi h_\eta [\varphi_{i,j,k_1} + \varphi_{i+1,j,k_1} + \varphi_{i,j+1,k_1} + \varphi_{i+1,j+1,k_1} \\ & \qquad + \varphi_{i,j,k_2} + \varphi_{i+1,j,k_2} + \varphi_{i,j+1,k_2} + \varphi_{i+1,j+1,k_2}] \\ & + \textstyle\sum_{k=k_1}^{k_2-1} \sum_{i=i_1}^{i_2-1} h_\xi h_\zeta [\varphi_{i,j_1,k} + \varphi_{i+1,j_1,k} + \varphi_{i,j_1,k+1} + \varphi_{i+1,j_1,k+1} \\ & \qquad + \varphi_{i,j_2,k} + \varphi_{i+1,j_2,k} + \varphi_{i,j_2,k+1} + \varphi_{i+1,j_2,k+1}] \\ & + \textstyle\sum_{k=k_1}^{k_2-1} \sum_{j=j_1}^{j_2-1} h_\eta h_\zeta [\varphi_{i_1,j,k} + \varphi_{i_1,j+1,k} + \varphi_{i_1,j,k+1} + \varphi_{i_1,j+1,k+1} \\ & \qquad + \varphi_{i_2,j,k} + \varphi_{i_2,j+1,k} + \varphi_{i_2,j,k+1} + \varphi_{i_2,j+1,k+1}]\} \end{aligned}$$

where i_1, i_2, j_1, j_2, k_1 and k_2 are specified constants, such that $1 < i_1 < i_2 < N_\xi$, $1 < j_1 < j_2 < N_\eta$ and $1 < k_1 < k_2 < N_\zeta$, and

$$\varphi = c_2\{u^{(5)} - 0.5(\frac{[u^{(1)}]^2 + [u^{(2)}]^2 + [u^{(3)}]^2)}{u^{(1)}})\}$$

The validity of each these quantities is measured according to

$$\frac{|X_c - X_r|}{|X_r|} \leq \epsilon$$

where X_c and X_r are the computed and reference values, respectively, and ϵ is the maximum allowable relative error. The value of ϵ is specified to be 10^{-8}. The values of X_r are dependent on the numerical scheme under consideration and the values specified for $N_\xi, N_\eta, N_\zeta, \Delta\tau$ and N_s.

3.5.2 Benchmark 1

Class A: Perform $N_s = 200$ iterations of the Approximate Factorization Algorithm, with the following parameter values:

$$N_\xi = 64; \qquad N_\eta = 64; \qquad N_\zeta = 64$$

and

$$\Delta\tau = 0.0008$$

Timing for this benchmark should begin just before the Step 1 of the first iteration is started and end just after the Step 5 of the N_s-th iteration is complete.

Class B: Same except $N_\xi = N_\eta = N_\zeta = 102$ and $\Delta\tau = 0.0003$.

3.5.3 Benchmark 2

Class A: Perform $N_s = 400$ iterations of the Diagonal Form of the Approximate Factorization Algorithm, with the following parameter values:

$$N_\xi = 64; \qquad N_\eta = 64; \qquad N_\zeta = 64$$

and

$$\Delta\tau = 0.0015$$

Timing for this benchmark should begin just before the Step 1 of the first iteration is started and end just after the Step 9 of the N_s-th iteration is complete.

Class B: Same except $N_\xi = N_\eta = N_\zeta = 102$ and $\Delta\tau = 0.001$.

3.5.4 Benchmark 3

Class A: Perform $N_s = 250$ iterations of the Symmetric Successive Over-Relaxation Algorithm with the following parameter values:

$$N_\xi = 64; \qquad N_\eta = 64; \qquad N_\zeta = 64$$

and

$$\Delta\tau = 2.0 \qquad \omega = 1.2$$

Timing for this benchmark should begin just before the Step 1 of the first iteration is started and end just after the Step 4 of the N_s-th iteration is complete.

Class B: Same except $N_\xi = N_\eta = N_\zeta = 102$.

For all benchmarks, values of the remaining constants are specified as

$$k_1 = 1.40; \qquad k_2 = 0.40; \qquad k_3 = 0.10; \qquad k_4 = 1.00; \qquad k_5 = 1.40$$

$$C_{1,1} = 2.0 \qquad C_{2,1} = 1.0 \qquad C_{3,1} = 2.0 \qquad C_{4,1} = 2.0 \qquad C_{5,1} = 5.0$$

$$C_{1,2} = 0.0 \qquad C_{2,2} = 0.0 \qquad C_{3,2} = 2.0 \qquad C_{4,2} = 2.0 \qquad C_{5,2} = 4.0$$

$$C_{1,3} = 0.0 \qquad C_{2,3} = 0.0 \qquad C_{3,3} = 0.0 \qquad C_{4,3} = 0.0 \qquad C_{5,3} = 3.0$$

$$C_{1,4} = 4.0 \qquad C_{2,4} = 0.0 \qquad C_{3,4} = 0.0 \qquad C_{4,4} = 0.0 \qquad C_{5,4} = 2.0$$

$$C_{1,5} = 5.0 \qquad C_{2,5} = 1.0 \qquad C_{3,5} = 0.0 \qquad C_{4,5} = 0.0 \qquad C_{5,5} = 0.1$$

$$C_{1,6} = 3.0 \qquad C_{2,6} = 2.0 \qquad C_{3,6} = 2.0 \qquad C_{4,6} = 2.0 \qquad C_{5,6} = 0.4$$

$$C_{1,7} = 0.5 \qquad C_{2,7} = 3.0 \qquad C_{3,7} = 3.0 \qquad C_{4,7} = 3.0 \qquad C_{5,7} = 0.3$$

$C_{1,8} = 0.02 \qquad C_{2,8} = 0.01 \qquad C_{3,8} = 0.04 \qquad C_{4,8} = 0.03 \qquad C_{5,8} = 0.05$

$C_{1,9} = 0.01 \qquad C_{2,9} = 0.03 \qquad C_{3,9} = 0.03 \qquad C_{4,9} = 0.05 \qquad C_{5,9} = 0.04$

$C_{1,10} = 0.03 \qquad C_{2,10} = 0.02 \qquad C_{3,10} = 0.05 \qquad C_{4,10} = 0.04 \qquad C_{5,10} = 0.03$

$C_{1,11} = 0.5 \qquad C_{2,11} = 0.4 \qquad C_{3,11} = 0.3 \qquad C_{4,11} = 0.2 \qquad C_{5,11} = 0.1$

$C_{1,12} = 0.4 \qquad C_{2,12} = 0.3 \qquad C_{3,12} = 0.5 \qquad C_{4,12} = 0.1 \qquad C_{5,12} = 0.3$

$C_{1,13} = 0.3 \qquad C_{2,13} = 0.5 \qquad C_{3,13} = 0.4 \qquad C_{4,13} = 0.3 \qquad C_{5,13} = 0.2$

$d_\xi^{(1)} = d_\xi^{(2)} = d_\xi^{(3)} = d_\xi^{(4)} = d_\xi^{(5)} = 0.75$

$d_\eta^{(1)} = d_\eta^{(2)} = d_\eta^{(3)} = d_\eta^{(4)} = d_\eta^{(5)} = 0.75$

$d_\zeta^{(1)} = d_\zeta^{(2)} = d_\zeta^{(3)} = d_\zeta^{(4)} = d_\zeta^{(5)} = 1.00$

$\varepsilon = [\max(d_\xi^{(1)}, d_\eta^{(1)}, d_\zeta^{(1)})]/4.0$

References

[1] Bailey, D. H.; and Barton, J. T.: The NAS Kernel Benchmark Program. NASA TM-86711, Aug. 1985.

[2] Pulliam, T. H.: Efficient Solution Methods for the Navier-Stokes Equations. Lecture Notes for the Von Karman Institute for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Jan. 20-24, 1986, Brussels, Belgium.

[3] Yoon. S.; Kwak, D.; and Chang, L.: LU-SGS Implicit Algorithm for Three-Dimensional Incompressible Navier-Stokes Equations with Source Term. AIAA Paper 89-1964-CP, 1989.

[4] Jameson, A.; Schmidt, W.; and Turkel, E.: Numerical Solution of the Euler Equations by Finite Volume Methods using Runge-Kutta Time-Stepping Schemes. AIAA Paper 81-1259, June 1981.

[5] Steger, J. L.; and Warming, R. F.: Flux Vector Splitting of the Inviscid Gas Dynamics Equations with Applications to Finite Difference Methods. Journal of Computational Physics, vol. 40, 1981, p. 263.

[6] van Leer, B.: Flux Vector Splitting for the Euler Equations. Eighth International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics, vol. 170, Edited by E. Krause, 1982, pp. 507-512.

[7] Roe, P. L.: Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. Journal of Computational Physics, vol. 43, 1981, pp. 357-372.

[8] Harten, A.: High Resolution Schemes for Hyperbolic Conservation Laws. Journal of Computational Physiscs, vol. 49, 1983, pp. 357-393.

[9] Gordon, W. J.: Blending Function Methods for Bivariate and Multivariate Interpolation. SIAM J. Num. Analysis, vol. 8, 1971, pp. 158.

[10] Warming, R. F.; and Beam, R. M.: On the Construction and Application of Implicit Factored Schemes for Conservation Laws. Symposium on Computational Fluid Dynamics, SIAM-AMS Proceedings, vol. 11, 1978.

[11] Lapidus, L.; and Seinfeld, J. H.: Numerical Solution of Ordinary Differential Equations. Academic Press, New York, 1971.

[12] Lomax, H.: Stable Implicit and Explicit Numerical Methods for Integrating Quasi-Linear Differential Equations with Parasitic-Stiff and Parasitic-Saddle Eigenvalues. NASA TN D-4703, 1968.

[13] Pulliam, T. H.; and Chaussee, D. S.: A Diagonal Form of an Implicit Approximate Factorization Algorithm. Journal of Computational Physics, vol. 39, 1981, p. 347.

[14] Chan, T .F. and Jackson, K. R.: Nonlinearly Preconditioned Krylov Subspace Methods for Discrete Newton Algorithms. SIAM J. Sci. Stat. Compt., vol. 5, 1984, pp. 533-542.

[15] Axelsson, O.: A Generalized SSOR Method. BIT, vol.13, 1972, pp. 443-467.

[16] Olsson, P.; and Johnson, S. L.: Boundary Modifications of the Dissipation Operators for the Three-Dimensional Euler Equations. Journal of Scientific Computing, vol. 4, 1989, pp. 159-195.

[17] Coakley, T. J.: Numerical Methods for Gas Dynamics Combining Characteristics and Conservation Concepts. AIAA Paper 81-1257, 1981.