Title
MANAGEMENT OF MULTIDIMENSIONAL DATA STRUCTURES IN NMR IMAGING

Authors
Veklerov, E.
Roos, M.S.
Mushlin, R.A.

Publication Date
1987-04-01

LBL-23353
Preprint

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

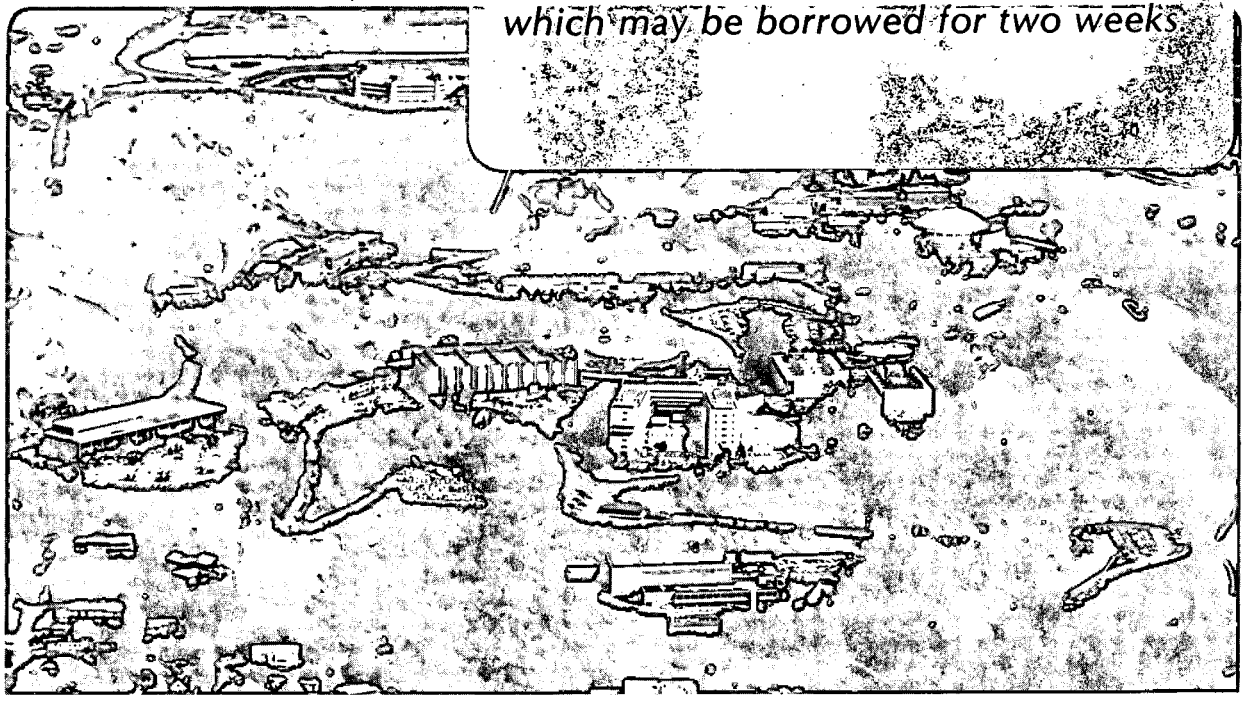
Engineering Division

Submitted to IEEE Transactions on
Software Engineering

MANAGEMENT OF MULTIDIMENSIONAL DATA STRUCTURES IN NMR IMAGING

E. Veklerov, M.S. Roos, and R.A. Mushlin

April 1987



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

LBL-23353

MANAGEMENT OF MULTIDIMENSIONAL DATA STRUCTURES IN NMR IMAGING

E. Veklerov and M.S. Roos
Lawrence Berkeley Laboratory
University of California
Berkeley CA 94720

R. A. Mushlin
IBM Corporation
Armonk, NY

This work was supported by the International Business Machines Corporation, Public Health Service Grant No. HL25840 awarded by the National Heart, Lung and Blood Institute, and the U.S. Department of Energy under Contract No. DE-AC03-76SF00098

Management of Multidimensional Data Structures in NMR Imaging[1]

Eugene Veklerov, Mark S. Roos

Lawrence Berkeley Laboratory

and

Richard A. Mushlin

IBM Corporation

29 April, 1987

Abstract

Multi-slice, multi-echo NMR imaging experiments involve handling large multidimensional data structures. Performance considerations require the capability to use these structures in memory efficiently. A set of commands for manipulation of data arrays is described. The commands use a special syntax to support vector operations. They have been implemented in the context of an instrument control and data analysis program.

Index Terms — NMR imaging, multidimensional data structures, vector processing, dynamic memory allocation.

[1]This work was supported by the International Business Machines Corporation, Public Health Service Grant No. HL25840 awarded by the National Heart, Lung and Blood Institute, and the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

1 Introduction

The problem of efficient handling of multidimensional data structures has arisen as part of a larger software project: the design and implementation of a real-time system to control NMR experiments and process their results. The system is implemented on an IBM S/9000 computer (IBM Corp., Armonk, NY) employing a Motorola 68000 microprocessor and augmented with a SKYMNK array processor (SKY Computers, Inc., Lowell, MA).

The scope of the project was determined by the character of its users. The system is intended to be used in a research environment by scientists and physicians who are familiar with programming. They may use the system at several levels. At the highest level, they may adopt available macros, each of which is a collection of commands performing a specific function. A library of macros has been prepared which covers most commonly used functions, such as initializing the spectrometer, processing signal data, displaying an image, etc.

However, it is envisioned that some users may wish to develop their own macros. To this end, the system provides a comprehensive set of commands, such as 'create a data buffer', 'add the contents of two buffers', 'perform the Fourier transform', etc. Macros are run under an executive which invokes corresponding subrou-

tines, passes parameters and provides error handling. The software is written in Pascal and 68000 assembly language. The hardware and software environment is described in more detail in [1]. The present paper concentrates on only one aspect of the system – management of multidimensional data structures.

Each echo sampled in an NMR imaging experiment represents a line in the two dimensional space that is related to the desired image by Fourier transform (K-space). The complete experiment consists of a loop sampling a rectangle in K-space in a raster scan fashion, the offset of a line being determined by a process called phase encoding. NMR data is simultaneously acquired for several slices through the object and for one or more echo times. The echo time determines the relative contribution of relaxation processes to the contrast of the image. A typical multi-slice, multi-echo experiment involves transfers of more than 10,000 complex words or 80,000 bytes per second into disk or memory based data structures as large as 2.5M words. This imposes stringent requirements on performance characteristics and flexibility of the system. In particular, it is necessary to be able to create data buffers in memory, delete them and subsequently re-allocate the memory for other purposes. Another requirement is the capability to view the data buffers as multidimensional entities and process them accordingly.

Moreover, it is desirable to provide the user with the following capabilities.

- To change the dimensionality of a buffer. For example, a multi-echo image data array is acquired as a two dimensional structure: the echo train from each excitation pulse sequence (and phase encoding gradient value) is stored as it is moved from a hardware averager. For processing, it is convenient to view the data as a four dimensional structure, so that an image may be reconstructed for each slice and echo.

- To transpose subscripts of a buffer or re-order its dimensions, so that columns may become rows and vice versa.

- To extract a 'sub-buffer' from a buffer. Thus, the user may wish to extract a view from a buffer containing reconstructions of images from several slices and echoes, to extract a single echo from the raw data, or to analyze a single row or column.

It is important to have these capabilities merely by changing buffer descriptors rather than by actual changing their contents. The following scheme is an efficient solution which meets all the above requirements and is easily incorporated into macros by the user of the system.

2 Buffer Access Scheme

Access to buffers is provided through buffer descriptors called buffer control blocks (BCB). The BCB is a record containing the buffer name, its type, dimensional information and the start address of the buffer. The system supports seven buffer types: byte, short integer, complex short integer, long integer, complex long integer, floating point and complex floating point. A buffer may have up to 5 dimensions and the dimensional information specifies the number of elements in each dimension beginning with the innermost one.

In order to create a buffer, the user issues a command like this:

ALLOC BIGBUF 'I 256 2 12 256

which attempts to allocate memory for a 4-dimensional buffer called *BIGBUF*. Its dimensions (beginning with the innermost one) are 256, 2, 12 and 256 and its type is short integer. Such a buffer may contain data from an NMR experiment with 256 samples per echo, 2 echoes, 12 slices and 256 phase encodings. *I* denotes the short integer type and the single quote before it means that the *I* is a literal rather than a variable passed to this command.

If enough memory is available, the command allocates space for this buffer from the heap and makes a BCB to manage it. When

a buffer is deleted, its memory space is released and its BCB is removed.

Suppose the user wants to store the contents of a buffer in a file. Files are viewed as multidimensional entities analogous to buffers. In order to create a file, the user may issue this command:

MAKEFILE 'SMALLFIL 'I 256 256

which will try to reserve disk space for a file of 256 by 256 short integers and make a file control block (FCB) which has the same structure as the BCB.

In order to write the contents of a buffer in a file, the user can use the *WRITE* command. It provides two capabilities:

- The source, the destination, or both can be subspaces of the memory buffer, the disk file, or both, respectively. This means that some subscripts can be fixed, while the others take values from the entire ranges defined in the BCBs and FCBs. One could imagine a more general scheme in which a subscript takes values from a subrange of the entire range. However, we have found that our scheme is adequate for most of NMR imaging experiments, and the added complexity of the subrange specification is not justified.

- BCBs and FCBs describe the corresponding objects in terms

of ordered lists: their innermost dimensions, the second innermost dimensions, etc. The *WRITE* command (and some others) support a more general view whereby these objects are simply multidimensional parallelepipeds and the user may wish to alter the order of their dimensions between the source and the destination.

Then the user may issue the following command:

WRITE BIGBUF *1 2 7 *2 *TO SMALLFIL* *1 *2

This command means that the second and the third subscripts of the source are fixed constants equal to 2 and 7, respectively. *1 and *2 in the first and fourth positions of the source mean that, as the contents of the source are transferred, the first subscript remains the most rapidly changing subscript and the fourth subscript is the second most rapidly changing one. *1 and *2 in the first and second positions of the destination mean that the transfered vectors are loaded in the natural order.

Let us describe the syntax in general. Both the source and the destination consist of a name followed by up to 5 dimensional parameters. A dimensional parameter is either a number or an asterisk followed by a number from 1 to 5. If the i-th position is occupied by a number, say N, then the i-th dimension of the corresponding buffer is fixed and equal to N. If it is occupied by

*k, then the i-th dimension of the corresponding buffer will be the k-th most rapidly changing subscript during this transfer. Trailing 1's may be omitted. For example, the command

*WRITE BIGBUF *1 TO SMALLFIL *1 7*

will be expanded to

*WRITE BIGBUF *1 1 1 1 1 TO SMALLFIL *1 7 1 1 1*

which means that the very first segment of 256 2-byte words will be extracted from *BIGBUF* and copied onto the 7-th segment of *SMALLFIL*.

Let us take another example. The dimensions of the source buffer, A, are 256, 4, 16. The destination file, B, is defined as 256, 2, 16, 4. Then the command

*WRITE A *1 *2 *3 TO B *1 1 *3 *2*

will transfer the data as follows. It will extract segments of 256 words from the source in their natural order but load them into the destination in such a way that the second subscript of the destination is fixed and equal to 1 while the fourth subscript of the destination varies faster than its third subscript.

The command *READ* has the same syntax as *WRITE* and it copies disk files into memory buffers. Additional flexibility is pro-

vided by the *EQUIV* (equivalence) command. This powerful command makes another BCB for an existing buffer. In other words, it generates another descriptor with a new name and new dimensional information pointing to an existing buffer. For example,

EQUIV OLDBUF TO NEWBUF 'I 256 4

introduces an additional new name, *NEWBUF*, for a buffer, *OLDBUF*, which already exists. However, if this buffer is referred to under the new name, its dimensions are assumed to be 256 by 4, whereas if it is referred to under its old name, its dimensions are treated as 256 by 2 by 2.

One buffer may have several BCBs. For instance,

ALLOC A 'R 128 4 8

EQUIV A TO B 'R 512 8

EQUIV B TO C 'R 4096

is a valid sequence of commands. Equivalent BCBs must reference the same total number of bytes but they may make different assumptions about the type of the referenced buffer. Thus, after these commands

ALLOC COMPLEXBUF 'X 256

EQUIV REALBUF 'R 2 256

the same buffer can be referred to under the name *COMPLEXBUF* assuming that it contains 256 complex floating point numbers (*X*

is the designation for this type) or under the name *REALBUF* with 2 by 256 real floating-point numbers (*R* is the code of this type). Having done that, the user may wish to store only the real or imaginary components of the buffer in a disk file, or operate on the components separately.

3 Buffer Commands

Buffer commands fall into two categories – buffer management commands and buffer operation commands.

A. Buffer Management Commands

Several commands of this category have already been mentioned: *ALLOC, MAKEFILE, WRITE, READ* and *EQUIV*. There are two commands to delete a buffer and a file: *BD* and *FD*. Two other commands, *BCAT* and *FCAT*, display the catalogs of all current buffers and files, respectively, complete with data type and dimension information.

The remaining four commands extract fields of BCBs and FCBs and assign them to variables which then may be passed as parameters to other commands within the same macro.

BTYP XBUF ABC

and

FTYP YFILE XYZ

assign the type of the buffer *XBUF* to the variable *ABC* and the type of the file *YFILE* to the variable *XYZ*.

BDIM SIGNAL 2 M2

and

FDIM DATA 3 N3

assign the second dimension of the buffer *SIGNAL* to the variable *M2* and the third dimension of the file *DATA* to the variable *N3*. These commands are useful for writing parameterized macros.

B. Buffer Operation Commands

There are over 50 commands operating on the contents of buffers. They all begin with a 'B' to distinguish them from their counterparts operating on variables. Most of the buffer operation commands employ the array processor.

The elements of a buffer participating in a command are specified by dimensional parameters. Each dimensional parameter is either a number (or a variable) or an asterisk. In the former case, the value of the dimension is fixed and equal to the given value. In the latter case, the entire range of this dimension participates in the operation.

Most of these commands have a simpler syntax than the *WRITE*

and *READ* commands. They do not allow the user to re-order subscripts using the *1, *2, etc. notation. If the user needs to transpose some subscripts of a buffer in memory, he may use the *BXP* command that has the same syntax as the *WRITE* and *READ* commands. As a result of this decision, the other operations on buffers use only fixed values or asterisks as their arguments. An asterisk denotes the entire range in the corresponding dimension and this notation is sufficient because the order of dimensions is preserved.

If two or more buffers participate in an operation, the compatibility of their dimensions is verified. Some operations may not be performed on buffers of some types. Thus, the operation of computing cosine values of buffer elements requires that the elements be real.

Let us consider an example. The command

*BADD FIRST * * AND SECOND * 3 * TO THIRD * *

means that elements of the buffer *FIRST* are added to elements of the buffer *SECOND* and the results become elements of the buffer *THIRD*. The second subscript of the buffer *SECOND* always equals 3. The inner subscript of this operation is the first subscript of all three buffers. The outer subscript is the second subscript of

FIRST, the third subscript of *SECOND* and the second subscript of *THIRD*. It is required that the ranges of the corresponding subscripts be the same for all three buffers.

As usual, if less than 5 dimensional arguments are specified, the remaining arguments are assumed to be 1's. Therefore,

BADD A AND B TO C

is equivalent to

BADD A 1 1 1 1 1 *AND B* 1 1 1 1 1 *TO C* 1 1 1 1 1

which is different from

BADD A * * * * * *AND B* * * * * * *TO C* * * * * *

Syntactically, the buffer operation commands can be broken into several classes. The remainder of this section consists of several examples of such commands which clarify their syntax.

Scalar-to-buffer commands

The simplest example of commands of this class is *BSET*. The command

BSET 3.14 *TO ABC* * 2

sets all elements of the second segment of *ABC* to 3.14.

Buffer-to-scalar commands

The command

*BSUM X * * TO TOTAL*

sums the specified elements of the buffer *X* and assigns the result to a variable called *TOTAL*.

Buffer-to-buffer commands

Many of these commands are elementary functions, such as

*BSIN ANGLE * TO ITSSIN * K*

which computes sine values. This operation may be performed in place. Another example is

*BFFT X * TO X **

which performs the Fast Fourier Transform of the source buffer and puts result in the destination buffer. It may also be performed in place. The type of the buffers must be complex floating-point and the innermost dimension must be a power of two for this operation.

Buffer-and-buffer-to-buffer commands

These commands use the contents of two buffers to produce another buffer.

*BMUL FACTOR1 * * AND FACTOR2 * * TO PROD * **

multiplies elements of the two source buffers and puts the products into the destination buffer.

Scalar-and-scalar-to-buffer commands

The command

BRAMP 0 *AND* 1 *TO LINEAR* *

generates a linear sequence in which the first scalar is the first element and the second scalar is the increment. If the dimensional parameters of the buffer include, say, two asterisks, the same ramp function is generated for every value of the second subscript.

Scalar-and-buffer-to-buffer commands

A representative of this group of commands is

BCLIP MAX AND SIGNAL * *TO CUTSIG* *

which sets all participating elements of the buffer *SIGNAL* exceeding *MAX* to *MAX* and those which are less than *-MAX* to *-MAX* and leaves the other elements unchanged. The result is put into the buffer *CUTSIG*.

Scalar-and-scalar-and-buffer-to-buffer commands

The command

BBC 64 *AND* 64 *AND A* * *TO B* *

performs a baseline correction. It computes the average value of the first 64 and last 64 elements of the source buffer, subtracts it from all participating elements of the source and puts the result into the specified elements of the destination.

Buffer-and-buffer-to-scalar commands

The command

*BDOT X * AND Y * TO VALUE*

computes the dot product of the two buffers and assigns the result to the scalar. If two or more asterisks are specified, all dot products computed within the innermost loops are summed up.

4 CONCLUSION

A management system for multidimensional data structures arising in NMR imaging experiments has been designed and implemented. The system uses a special syntax allowing the user to visualize the multidimensional nature of data. A comprehensive set of commands is provided for the user who wishes to develop new macros. The overhead associated with the management system is negligible compared to the time spent in the array processor.

References

[1] M. S. Roos, R. A. Mushlin, E. Veklerov, J. D. Port, C. Ladd, C. G. Harrison. An Instrument Control and Data Analysis for Imaging and *in Vivo* Spectroscopy. 5th Annual Meeting, Society for Magnetic Resonance in Medicine, Montreal, Canada, 1986. *Book of Abstracts*, pp. 1127 – 1128. Manuscript in preparation.

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720