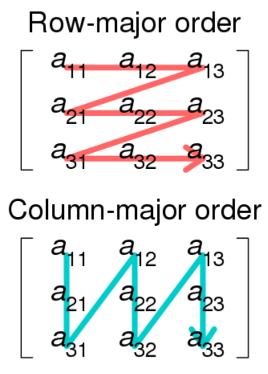
WIKIPEDIA

Row- and column-major order

In computing, **row-major order** and **column-major order** are methods for storing multidimensional arrays in linear storage such as random access memory.


Row-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
Column-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
Illustration of difference between row- and column-major ordering

The difference between the orders lies in which elements of an array are contiguous in memory. In a row-major order, the consecutive elements of a row reside next to each other, whereas the same holds true for consecutive elements of a column in a column-major order. While the terms allude to the rows and columns of a two-dimensional array, i.e. a matrix, the orders can be generalized to arrays of any dimension by noting that the terms row-major and column-major are equivalent to lexicographic and colexicographic orders, respectively.

Data layout is critical for correctly passing arrays between programs written in different programming languages. It is also important for performance when traversing an array because modern CPUs process sequential data more efficiently than nonsequential data. This is primarily due to CPU caching. In addition, contiguous access makes it possible to use SIMD instructions that operate on vectors of data. In some media such as tape or NAND flash memory, accessing sequentially is orders of magnitude faster than nonsequential access.

Contents

Explanation and example

The terms row-major and column-major stem from the terminology related to ordering objects. A general way to order objects with many attributes is to first group and order them by one attribute, and then, within each such group, group and order them by another attribute, etc. If more than one attribute participate in ordering, the first would be called *major* and the last *minor*. If two attributes participate in ordering, it is sufficient to name only the major attribute.

In the case of arrays, the attributes are the indices along each dimension. For matrices in mathematical notation, the first index indicates the *row*, and the second indicates the *column*, e.g., given a matrix A, $a_{1,2}$ is in its first row and second column. This convention is carried over to the syntax in programming languages,[1] although often with indexes starting at 0 instead of 1.[2]

Even though the row is indicated by the *first* index and the column by the *second* index, no grouping order between the dimensions is implied by this. The choice of how to group and order the indices, either by row-major or column-major methods, is thus a matter of convention. The same terminology can be applied to even higher dimensional arrays. Row-major grouping starts from the *leftmost* index and column-major from the *rightmost* index, leading to lexicographic and colexicographic (or colex) orders, respectively.

For example, for the array

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

the two possible ways (in C, indexing from 0) are

Row-major order (lexographical access order)

Address	Access	Value

Column-major order (colexographical access order)

Address	Access	Value

0	`A[0][0]`	a_{11}
1	`A[0][1]`	a_{12}
2	`A[0][2]`	a_{13}
3	`A[1][0]`	a_{21}
4	`A[1][1]`	a_{22}
5	`A[1][2]`	a_{23}

0	`A[0][0]`	a_{11}
1	`A[1][0]`	a_{21}
2	`A[0][1]`	a_{12}
3	`A[1][1]`	a_{22}
4	`A[0][2]`	a_{13}
5	`A[1][2]`	a_{23}

The two possible ways (from Fortran, 1-indexed):

Row-major order (colexographical access order)

Address	Access	Value
1	`A(1,1)`	a_{11}
2	`A(1,2)`	a_{12}
3	`A(1,3)`	a_{13}
4	`A(2,1)`	a_{21}
5	`A(2,2)`	a_{22}
6	`A(2,3)`	a_{23}

Column-major order (lexographical access order)

Address	Access	Value
1	`A(1,1)`	a_{11}
2	`A(2,1)`	a_{21}
3	`A(1,2)`	a_{12}
4	`A(2,2)`	a_{22}
5	`A(1,3)`	a_{13}
6	`A(2,3)`	a_{23}

Note how the use of `A[i][j]` with multi-step indexing as in C, as opposed to a neutral notation like `A(i,j)` as in Fortran, almost inevitably implies row-major order for syntactic reasons, so to say, because it can be rewritten as `(A[i])[j]`, and the `A[i]` row part can even be assigned to an intermediate variable that is then indexed in a separate expression. (No other implications should be assumed, e.g., Fortran is not column-major simply *because* of its notation, and even the above implication could intentionally be circumvented in a new language.)

To use column-major order in a row-major environment, or vice versa, for whatever reason, one workaround is to assign non-conventional roles to the indexes (using the first index for the column and the second index for the row), and another is to bypass language syntax by explicitly computing positions in a one-dimensional array. Of course, deviating from convention probably incurs a cost that increases with the degree of necessary interaction with conventional language features and other code, not only in the form of increased vulnerability to mistakes (forgetting to also invert matrix multiplication order, reverting to convention during code maintenance, etc.), but also in the form of having to actively rearrange elements, all of which have to be weighed against any original purpose such as increasing performance.

Programming languages and libraries

Programming languages or their standard libraries that support multi-dimensional arrays typically have a native row-major or column-major storage order for these arrays.

Row-major order is used in C/C++/Objective-C (for C-style arrays), PL/I,[3] Pascal,[4] Speakeasy, SAS,[5] and Rasdaman.[6]

Column-major order is used in Fortran, MATLAB,[7] GNU Octave, S-Plus,[8] R,[9] Julia,[10] and Scilab.[11]

A special case would be OpenGL (and OpenGL ES) for graphics processing. Since "recent mathematical treatments of linear algebra and related fields invariably treat vectors as columns," designer Mark Segal decided to substitute this for the convention in predecessor IRIS GL, which was to write vectors as rows; for compatibility, transformation matrices would still be stored in vector-major rather than coordinate-major order, and he then used the "subterfuge [to] say that matrices in OpenGL are stored in column major order".[12] This was really only relevant for presentation, because matrix multiplication was stack-based and could still be interpreted as post-multiplication, but, worse, reality leaked through the C-based API because individual elements would be accessed as `M[vector][coordinate]` or, effectively, `M[column][row]`, which unfortunately muddled the convention that the designer sought to adopt, and this was even preserved in the OpenGL Shading Language that was later added (although this also makes it possible to access coordinates by name instead, e.g., `M[vector].y`). As a result, many developers will now simply declare that having the column as the first index is the definition of column-major, even though this is clearly not the case with a real column-major language like Fortran.

Neither row-major nor column-major

A typical alternative for dense array storage is to use Iliffe vectors, which typically store elements in the same row contiguously (like row-major order), but not the rows themselves. They are used in (ordered by age): Java,[13] C#/CLI/.Net, Scala,[14] and Swift.

Even less dense is to use lists of lists, e.g., in Python,[15] and in the Wolfram Language of Wolfram Mathematica.[16]

An alternative approach uses tables of tables, e.g., in Lua.[17]

External libraries

Support for multi-dimensional arrays may also be provided by external libraries, which may even support arbitrary orderings, where each dimension has a stride value, and row-major or column-major are just two possible resulting interpretations.

Row-major order is the default in NumPy[18] (for Python).

Column-major order is the default in Eigen[19] (for C++).

Torch (for Lua) changed from column-major[20] to row-major[21] default order.

Transposition

As exchanging the indices of an array is the essence of array transposition, an array stored as row-major but read as column-major (or vice versa) will appear transposed. As actually performing this rearrangement in memory is typically an expensive operation, some systems provide options to specify individual matrices as being stored transposed. The programmer must then decide whether or not to rearrange the elements in memory, based on the actual usage (including the number of times that the array is reused in a computation).

For example, the Basic Linear Algebra Subprograms functions are passed flags indicating which arrays are transposed.[22]

Address calculation in general

The concept generalizes to arrays with more than two dimensions.

For a d-dimensional $N_1 \times N_2 \times \cdots \times N_d$ array with dimensions N_k (k=1...d), a given element of this array is specified by a tuple $(n_1, n_2, \ldots, n_d)$ of d (zero-based) indices $n_k \in [0, N_k - 1]$.

In row-major order, the *last* dimension is contiguous, so that the memory-offset of this element is given by:

$$n_d + N_d \cdot (n_{d-1} + N_{d-1} \cdot (n_{d-2} + N_{d-2} \cdot (\cdots + N_2 n_1)\cdots))) = \sum_{k=1}^{d} \left(\prod_{\ell=k+1}^{d} N_\ell \right) n_k$$

In column-major order, the *first* dimension is contiguous, so that the memory-offset of this element is given by:

$$n_1 + N_1 \cdot (n_2 + N_2 \cdot (n_3 + N_3 \cdot (\cdots + N_{d-1} n_d)\cdots))) = \sum_{k=1}^{d} \left(\prod_{\ell=1}^{k-1} N_\ell \right) n_k$$

where the empty product is the multiplicative identity element, i.e., $\prod_{\ell=1}^{0} N_\ell = \prod_{\ell=d+1}^{d} N_\ell = 1$.

For a given order, the stride in dimension k is given by the multiplication value in parentheses before index n_k in the right hand-side summations above.

More generally, there are $d!$ possible orders for a given array, one for each permutation of dimensions (with row-major and column-order just 2 special cases), although the lists of stride values are not necessarily permutations of each other, e.g., in the 2-by-3 example above, the strides are (3,1) for row-major and (1,2) for column-major.

See also

- Array data structure
- Matrix representation
- Vectorization (mathematics), the equivalent of turning a matrix into the corresponding column-major vector
- CSR format, a technique for storing sparse matrices in memory
- Morton order, another way of mapping multidimensional data to a one-dimensional index, useful in tree data structures

References

1. "Arrays and Formatted I/O" (http://www.fortrantutorial.com/arrays-formatted-io/index.php). *FORTRAN Tutorial*. Retrieved 19 November 2016.

2. "Why numbering should start at zero" (https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html). *E. W. Dijkstra Archive*. Retrieved 2 February 2017.
3. "Language Reference Version 4 Release 3" (https://www.ibm.com/support/knowledgecenter/SSQ2R2_9.0.0/com.ibm.ent.pl1.zos.doc/topics/lrm.pdf) (PDF). IBM. Retrieved 13 November 2017. "Initial values specified for an array are assigned to successive elements of the array in row-major order (final subscript varying most rapidly)."
4. "ISO/IEC 7185:1990(E)" (http://www.pascal-central.com/docs/iso7185.pdf) (PDF). "An array-type that specifies a sequence of two or more index-types shall be an abbreviated notation for an array-type specified to have as its index-type the first index-type in the sequence and to have a component-type that is an array-type specifying the sequence of index-types without the first index-type in the sequence and specifying the same component-type as the original specification."
5. "SAS® 9.4 Language Reference: Concepts, Sixth Edition" (http://documentation.sas.com/api/docsets/lrcon/9.4/content/lrcon.pdf) (PDF). SAS Institute Inc. September 6, 2017. p. 573. Retrieved 18 November 2017. "From right to left, the rightmost dimension represents columns; the next dimension represents rows. [...] SAS places variables into a multidimensional array by filling all rows in order, beginning at the upper left corner of the array (known as row-major order)."
6. "Internal array representation in rasdaman" (http://rasdaman.org/wiki/RasdamanInternalArrayRepresentation). *rasdaman.org*. Retrieved 8 June 2017.
7. MATLAB documentation, MATLAB Data Storage (http://www.mathworks.co.uk/help/matlab/matlab_external/matlab-data.html#f22019) (retrieved from Mathworks.co.uk, January 2014).
8. Spiegelhalter et al. (2003, p. 17): Spiegelhalter, David; Thomas, Andrew; Best, Nicky; Lunn, Dave (January 2003), "Formatting of data: S-Plus format", *WinBUGS User Manual* (https://web.archive.org/web/20120303024931/http://www.mrc-bsu.cam.ac.uk/bugs/) (Version 1.4 ed.), Robinson Way, Cambridge CB2 2SR, UK: MRC Biostatistics Unit, Institute of Public Health, PDF document (http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/manual14.pdf), archived from the original (http://www.mrc-bsu.cam.ac.uk/bugs) on 2012-03-03
9. *An Introduction to R*, Section 5.1: Arrays (https://cran.r-project.org/doc/manuals/R-intro.html#Arrays) (retrieved March 2010).
10. "Multi-dimensional Arrays" (http://docs.julialang.org/en/release-0.4/manual/arrays/). *Julia*. Retrieved 6 February 2016.
11. "FFTs with multidimensional data" (https://wiki.scilab.org/howto/FFTMultidimensionalData). *Scilab Wiki*. Retrieved 25 November 2017. "Because Scilab stores arrays in column major format, the elements of a column are adjacent (i.e. a separation of 1) in linear format."
12. "Column Vectors Vs. Row Vectors" (http://steve.hollasch.net/cgindex/math/matrix/column-vec.html). Retrieved 12 November 2017.
13. "Java Language Specification" (https://docs.oracle.com/javase/specs/jls/se7/html/jls-10.html). Oracle. Retrieved 13 February 2016.
14. "object Array" (http://www.scala-lang.org/api/current/#scala.Array$). *Scala Standard Library*. Retrieved 1 May 2016.
15. "The Python Standard Library: 8. Data Types" (https://docs.python.org/3.6/library/datatypes.html). Retrieved 18 November 2017.
16. "Vectors and Matrices" (http://reference.wolfram.com/language/tutorial/VectorsAndMatrices.html). *Wolfram*. Retrieved 12 November 2017.
17. "11.2 – Matrices and Multi-Dimensional Arrays" (http://www.lua.org/pil/11.2.html). Retrieved 6 February 2016.
18. "The N-dimensional array (ndarray)" (http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html). *SciPy.org*. Retrieved 3 April 2016.
19. "Eigen: Storage orders" (https://eigen.tuxfamily.org/dox/group__TopicStorageOrders.html). *eigen.tuxfamily.org*. Retrieved 2017-11-23. "If the storage order is not specified, then Eigen defaults to storing the entry in column-major."
20. "Tensor" (http://torch5.sourceforge.net/manual/torch/index-6.html). Retrieved 6 February 2016.
21. "Tensor" (https://github.com/torch/torch7/blob/master/doc/tensor.md#storage-storage). *Torch Package Reference Manual*. Retrieved 8 May 2016.
22. "BLAS (Basic Linear Algebra Subprograms)" (http://www.netlib.org/blas/). Retrieved 2015-05-16.

Sources

- Donald E. Knuth, *The Art of Computer Programming Volume 1: Fundamental Algorithms*, third edition, section 2.2.6 (Addison-Wesley: New York, 1997).