BIT-WISE BASED MULTI-DIMENSIONAL ARRAY

HP INC

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
HP INC, "BIT-WISE BASED MULTI-DIMENSIONAL ARRAY", Technical Disclosure Commons, (July 30, 2018)
https://www.tdcommons.org/dpubs_series/1379
**Bit-Wise Based Multi-Dimensional Array**

A method for generating large .Net arrays which are bigger than 2GB (.Net size limitation) and are performance oriented is disclosed. The method involves using multiple arrays and bit-wise operations for array indexing in order to create a new form of a container which functions as array with 64 bit indexer (as opposed to 32 bit indexing of .Net arrays), based on multiple dimensions.

Currently, .Net arrays use an indexer of Int32 which then limits the entire array size up to 2 GB of data:

Maximal value of int32 = 2,147,483,648.

As a result, existing software programs which use .Net arrays can’t support data which can be larger than 2 GB of data in a single array, and if your program is suddenly facing a larger quantities of data, you should either change your architecture to support segments of this data, in blocks limited to 2GB or to use an abstraction layer of a single array, with a 64 indexer, containing multiple inner arrays which then uses division and modulo operations for calculating both indexes of selected inner array and their indexed locations inside those inner arrays.

This is considered and the most common solution for this issue:

![The common solution – using continues multiple arrays](image)

Fig. #1 - The Div/Mod common solution

As written above, although this is a solution which quite simple to implement, its overall performance rating may not be acceptable in many, as it can deteriorate the overall performance of the program involved.

This disclosure, however, solves this exact performance issue, while trying to enlarge .Net array size and use an abstraction layer for an array, for backwards competitiveness with existing code.

It deals with assuring fast memory access to the array cells, getting into the memory segments in O(1) complexity, as with primitive arrays. This can be accomplished by using Bitwise operations, which are the fastest CPU operations available.

The solution suggests using 2D/4D bitwise based array, combined of several primitive .Net arrays.
2D case: 2 arrays, should cover up to 4GB

4D case: 4 arrays, should cover up to 8GB

Solution - Bitwise based multi dimensional array

- Motivation
  - Fast calculation of array index and the index inside the array
- Bitwise operations can do the trick (Opposite to div/mod operations which are very pure performance oriented)
  - Array selected = index & 1
  - Index in array = index >> 1
- For instance
  - index 5 = 5 & 1 = 1, 5 >> 1 = 2 (2nd array, 2nd place)
  - index 8 = 8 & 1 = 0, 8 >> 1 = 4 (1st array, 4th place)

Testing the solution on big arrays sizes showed remarkable performance results, pretty much as using a single .Net primitive array. The ‘set’ operation is a bit slower but still in the same performance range. The ‘get’ operation is as fast as a simple ‘get’ operation of any other primitive array.

In the majority of the cases, where 4GB sizes (twice as much the limitation we have today), are required, a 2D array will be sufficient and therefore only 1 bit operation will be used.

In addition, iterations on segments in memory will also be performance oriented (as with primitive arrays). It will use a nested loop which use dual indexes, the first index as the index inside the internal array and second one is the selected internal array itself. Both will be advancing side by side while iterating on the memory segment range of the multi dimensional array.

*Disclosed by Asa Bornstein, HP Inc.*