Large array arithmetics in C#

Asked 16 years, 3 months ago Modified 16 years, 3 months ago

Viewed 11k times

10

Which is the best way to store a 2D array in c# in order to optimize performance when performing lots of arithmetic on the elements in the array?

We have large (approx 1.5G) arrays, which for example we want to multiply with each other element by element. Performance is critical. The context in which this is done is in c#. Is there any smart way of storing the arrays and iterating over them? Could we write these parts in unmanaged C++ and will this really increase performance? The arrays need to be accessible to the rest of the c# program.

Currently (in c) the array is stored as a single long vector. We perform calculations on each element in the array and overwrite the old value. The calculations are usually unique for each element in the vector.

Timing experiments show that storing and iterating over the data as an array in C# is slower than storing it as a 2D array. I would like to know if there is an even better way of handling the data. The specific arithmetics performed are not relevant for the question.

c# arrays

Share

Improve this question

Follow

edited Sep 21, 2008 at 15:14

Kev
120k ● 53 ● 305 ● 391

asked Sep 21, 2008 at 13:33

AnnaR
3,186 ● 6 ● 36 ● 39

I'm not sure what you mean by the bit about your timing experiments. Do you mean that the C# 2D array was slower than the C 2D array? – Cameron MacFarland Sep 21, 2008 at 14:17

The c# 2d array was quicker than the 1d c# array – AnnaR Sep 21, 2008 at 16:05

4 Answers

Sorted by: Highest score (default)

8

Anna,

Here is a great page that discusses the performance difference between tradition scientific programming languages (fortran, C++) and c#.

http://msdn.microsoft.com/en-us/magazine/cc163995.aspx

According to the article C#, when using rectangular arrays (2d) can be a very good performer. Here is a graph that shows the difference in performance between jagged arrays (an array of arrays) and rectangular arrays (multi-dimensional) arrays.

[alt text http://i.msdn.microsoft.com/cc163995.fig08.gif](http://i.msdn.microsoft.com/cc163995.fig08.gif)

I would suggest experimenting yourself, and use the Performance Analysis in VS 2008 to compare.

If using C# is "fast enough" then your application will be that much easier to maintain.

Good Luck!

Share Improve this answer Follow

answered Sep 21, 2008 at 13:55
Jason Stevenson
3,990 ● 3 ● 32 ● 49

2 The results shown in that graph are from .Net 1.1. Jagged arrays are actually faster than multidimensional arrays in .Net 3 and 4 as shown here: codeproject.com/KB/cross-platform/BenchmarkCppVsDotNet.aspx – AndrewS Sep 15, 2011 at 12:07

1 @AndrewS - .Net 4 wasn't released when I answered this question back in 2008. – Jason Stevenson Sep 16, 2011 at 19:26

5

For best array performance, make sure you're using a single dimension array with lower index of 0.

To access the elements of the array as fast as possible, you can use unsafe pointers like so:

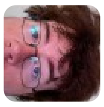
```
int[] array = Enumerable.Range(0, 1000).ToArray();

int count = 0;
unsafe {
    fixed (int* pArray = array) {
        for (int i = 0; i < array.Length; i++) {
            count += *(pArray + i);
        }
    }
}
```

EDIT Drat! Didn't notice you said 2D array. This trick won't work with a multi-dimensional array so I'm not sure how much help it will be. Although you could turn any array into a single-dimension array by doing some arithmetic on the array index. Just depends on if you care about the performance hit in indexing the array or in iterating over the array.

Share Improve this answer Follow

answered Sep 21, 2008 at 13:46



Cameron MacFarland
71.8k ● 20 ● 105 ● 134

1 @Cameron MacFarland: actually it can work for 2D arrays, you just have to do something like *(pArray + (ii * cols) + jj). – user7116 Sep 21, 2008 at 13:53

sixlettervariables: Good to know. Although this only applies to rectangular arrays [,] and not jagged arrays [][] unless you use the column count for the current array, which would make things slower. – Cameron MacFarland Sep 21, 2008 at 14:02

1 I suggest to measure, measure and ... ar least measure! Even if pointer arithmetic did bring a lot of performance gain in earlier versions of JIT, the optimization did advance a lot.

By now (.NET 4.0) the advantage seems to be almost negligible. – user492238 Apr 16, 2011 at 17:26

2

If you download F#, and reference one of the runtime libraries (I think it's FSharp.PowerPack), and use Microsoft.FSharp.Maths.Matrix. It optimises itself based on whether you are using a dense or sparse matrix.

Share Improve this answer Follow

answered Sep 21, 2008 at 14:03

TraumaPony

10.8k ● 13 ● 57 ● 75

0

Do you iterate the matrix by row or by colum or both? Do you always access nearby elements or do you do random accesses on the matrix.

If there is some locality in your accesses but you're not accessing it sequential (typical in matrix multiplication for example) then you can get a *huge* performance difference by storing your matrix in a more cache-friendly way.

A pretty easy way to do that is to write a little access function to turn your row/colum indices into an index and work on a one dimensional matrix, the cache-friendy way.

The function should group nearby coordinates into nearby indices. The morton-order can be used if you work on power of two sizes. For non-power sizes you can often bring just the lowest 4 bits into morton order and use normal index-arithmetic for the upper bits. You'll still get a significant speed-up, even if the coordinate to index conversion looks seems to be a costly operation.

http://en.wikipedia.org/wiki/Z-order_(curve) <-- sorry, can't link that SO does not like URL's with a dash in it. You have to cut'n'paste.

A speed up of factor 10 and more are realistic btw. It depends on the algorithm you ron over your matrices though.