Some weeks ago me and my friend was involved in a performance issue of an Image processing application which we wrote in C. We narrowed the bottleneck method. This method has bunch of pixel value multiplications to produce different colors. We improved the code but even that was not showing enough improvement to end user. We shared this problem with friends who work in IT industry but no one was to provide further improvement in that method. We were about to give up but recently one of them pointed us in right direction.

He told that “there is no scope of performance improvement in terms CPU instructions and we should look at how much time CPU is taking to read and write memory” and he was absolutely right.

Each multiplication takes 2-4 clock cycle while we waste 10-15 cycles (depends on CPU) if CPU fails to read the from L1 cache and wastage increases to 50-100 cycles if CPU fails to read the data from L2 cache and tries to read it L3 cache. Performance degrade even more if the data we are requesting is not present in L3 and now we try to read it from DRAM, it is the worst case scenario which we saw in our program. So simple multiplication which can be executed in ~5 clock cycles is now taking hundreds of clock cycle. After using Data locality and reducing false sharing we were able to increase the performance by 37%.

# Now what is CPU cache and what are cache lines?

The fastest memory structures are registers where CPU can read memory within 1 clock cycle but even modern CPUs such as intel i7 only has 8 and 16 registers for x86 and x64 architecture respectively and they can hold some thousand byte memory. These registers are not going to increase as they are super costly but on the other side of memory spectrum there is RAM which is extremely cheaper when compared to registers but tradeoff is speed. It takes around 1 clock cycle to read data from register while it takes hundreds of cycle to read it from DRAM. So to bridge this gap cache lines L1, L2 and L3 were created (There could be more, It totally depends on CPU architecture).

When you request data from particular memory CPU don’t only works for that request. It will try to predict what memory you will try to access in next request and place them in its cache lines. Now if its prediction is correct, you will get amazing data read/write speed but if its prediction is wrong and requested data is not present in cache line, performance will degrade. Generally according to the concept of data locality, CPU reads all the adjacent memory of your requested data and places it in CPU cache line to cater your next memory access request.

Three terms which you will frequently here in the world of CPU caching are:

**Cache hit:** Every time when CPU is able to find requested data in its cache line, it’s called cache hit.

**Cache miss:** Every time when CPU is not able to find data in given cache line, it’s called cache miss for that cache line. Your end goal while doing such performance improvement should be to decrease the value of cache miss.

**Data Locality:** In simplest terms keep your most important data (The data which you are going to access pretty frequently one after another) close to each other.

Here is an example of code with locality:

var sum = 0;

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

sum += a[i][j];

}

}

return sum;

Above example contains temporal locality because sum is accessed frequently in the loop. Temporal locality is exploited by keeping recently used instruction and data values in cache memory and by exploiting a cache hierarchy. Or even in a register, not in memory at all.

It also contains spatial locality because we have an array 'a' and we access each element of the array in order. Spatial locality is generally exploited by using larger cache blocks and by incorporating prefetching mechanisms (fetching items of anticipated use) into the cache control logic.

For more information on data locality look at this link:<https://medium.com/@adamzerner/spatial-and-temporal-locality-for-dummies-b080f2799dd>

Although most of the study which I did to solve my problem was in C/C++ code but I can demonstrate the same caching phenomenon in C# also.

Consider below use cases where arr1 and arr2 are int arrays with same length.

private static void UseCase1()

{

watch.Restart();

for (int i = 0; i < length; i++)

{

arr1[i] \*= 3;

arr2[i] \*= 3;

}

watch.Stop();

Console.WriteLine($"Usecase 1: {watch.ElapsedMilliseconds.ToString()} ms");

}

private static void UseCase2()

{

watch.Restart();

for (int i = 0; i < length; i++)

{

arr1[i] \*= 3;

}

for (int i = 0; i < length; i++)

{

arr2[i] \*= 3;

}

watch.Stop();

Console.WriteLine($"Usecase 2: {watch.ElapsedMilliseconds.ToString()} ms");

}

When you see UseCase2() what’s the first thing comes to your mind. Why the hell you are having two for loops with same limit and condition, you could improve your code...And I did. It became UseCase1(). Now tell me which will perform better. If you understood the concept of data locality, you guessed it right. UseCase2 will always perform better than UseCase1() And it performed 42% better than UseCase1() and this performance gap will increase when the size of array increases.

C:\Users\Ashutosh\Desktop\Pic1.JPG

# But why?

**UseCase2:**

When we requested arr1[0], CPU already had next 32 KB data in L1, 256KB data in L2 cache and 3MB in L3 cache for my machine. So as my collection is array where data is stored adjacent to each other most of my arr1 data is in cache lines and it will take less time to read that data and perform operation on it compare to reading it from RAM.So ultimately low cache miss.

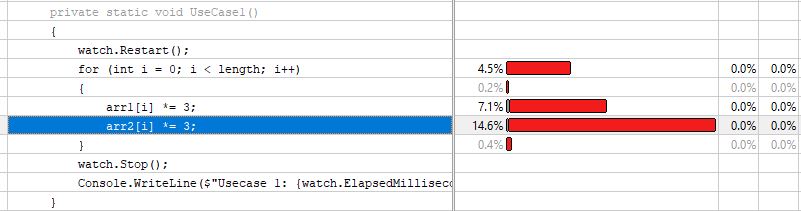
**UseCase1:**

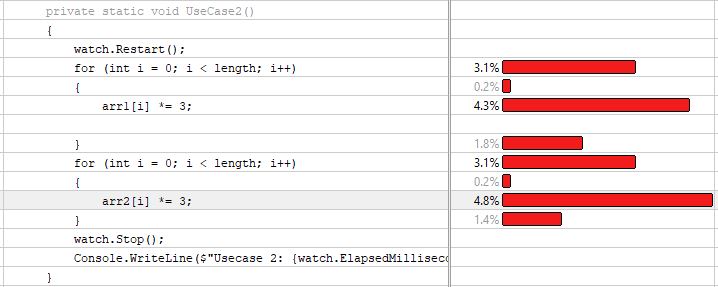
When we requested arr1[0], adjacent bytes will be stored in cache but now you ask for arr2[0] and CPU won’t find this data in cache and it has to read it from RAM and it will again build the cache lines.This will happen in each iteration of loop and this will cause multiple cache miss scenario which is why UseCase1 will always be slower.

**Profiler confirms the same:**

You can see that each time arr2 is accessed CPU takes more time compared to arr1 while in UseCase2 data access time for both the arrays are nearly same.

When you compare profiler data of usecase1 and usecase2, you can clearly see that cache miss cost easily out shadow the cost of extra loop and even the operation which I am performing in that loop which is why in such cases you should only use one collection in loop at a time and this only make sense when data is arranged sequentially in collection. This is main reason why Vector is much faster than Linked List.





Laying out the data in collection in such a way that they come next to each other in memory is all I wanted to do solve my problem and beware that I am not taking about references/pointers here, I am talking about data and this is the huge topic as it will cover all data structure concept in it and I have not even touched instruction caching. You know that code you write get converted to instruction and it also need to be stored somewhere before execution. If want to read about it someone wrote entire book on it. <http://www.dataorienteddesign.com/dodmain/>

Well as I said that the data you want to access should come together in cache lines, so data access take less CPU cycles but this statement only stands true when we are talking about one thread. If two threads tries to read/write data from same cache line it will be invalidate the cache line and will force cache miss scenario and this is also one form of false sharing.

Consider two variables within two different threads running on different cores (my fantasy of concurrency) are trying to access single data resource.

Imagine two different variables are being used by two different threads on two different cores. This appears to be embarrassingly parallel, as the different threads are using isolated data. However, if the two variables are located in the same cache line and at least one is being written, then there will be contention for that cache line. This is [false sharing](https://en.wikipedia.org/wiki/False_sharing).

It is called false sharing because even though the different threads are not sharing data, they are, unintentionally, sharing a cache line.

## My philosophy:

After reading this do I want you guys to break your massive for loop into multiple for loop where body will contain only one data collection? Of Course not, I want you to keep this information somewhere cached in your brain just in case you need it some day☺

Should you consider Cache optimization as a part of micro-optimization? Answer is it depends. Profile you code if cache miss cost is more than the operation you are trying to perform, it’s not a micro-optimization but let’s say if decreasing data access time to 0(this simulated in profilers) just give you improvement percent under 5%, it’s not worth it.

Many programmers think that writing cache friendly code is not something that they have to worry about and compilers do this job for them. Well, I don’t know how much and which segment of code is going to get optimized by compiler in which fashion. I rarely see my code in ILdasm. I don’t know exactly how compilers functions when Optimized code is enabled (holds true for csc as well as gcc). It may or may not optimized my code and I don’t like negative testing when I am dealing with third party software which I don’t understand, so I better write my code where it falls in happy day flows for compilers. So I don’t have to worry even if complier is not able to optimize it. I want to write more on this topic but I am not even sure if someone is going to read it or not…Even someone will try to read it but they will surely not read it up to this point☺ …So let’s wrap it up.

**Tools for such profiling:**

I use intel VTune (paid)

I heard cachegrind is also a good option (free)

**References:**

If you learn better by watching videos:

<https://www.youtube.com/watch?v=WDIkqP4JbkE&t=200s>

<https://www.youtube.com/watch?v=BP6NxVxDQIs&t=469s>

If you like to read articles:

<http://igoro.com/archive/gallery-of-processor-cache-effects/>

<https://lwn.net/Articles/255364/>

<https://en.wikipedia.org/wiki/Locality_of_reference>

<https://stackoverflow.com/questions/3928995/how-do-cache-lines-work>

<https://akkadia.org/drepper/cpumemory.pdf>

If have interest or want to discuss this topic, feel free to reach out to me.