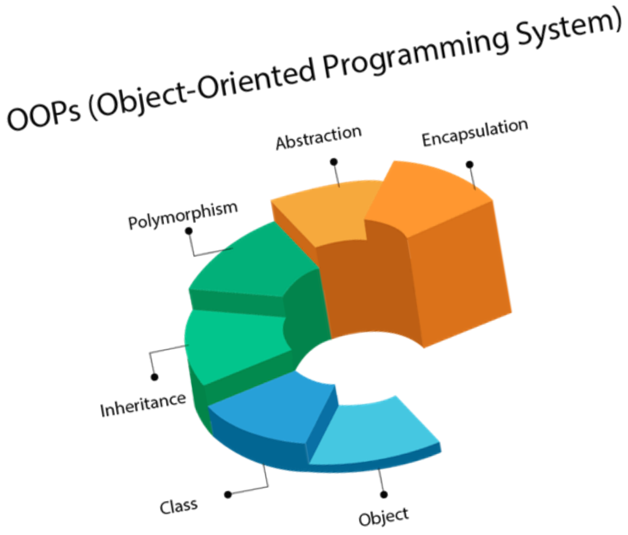
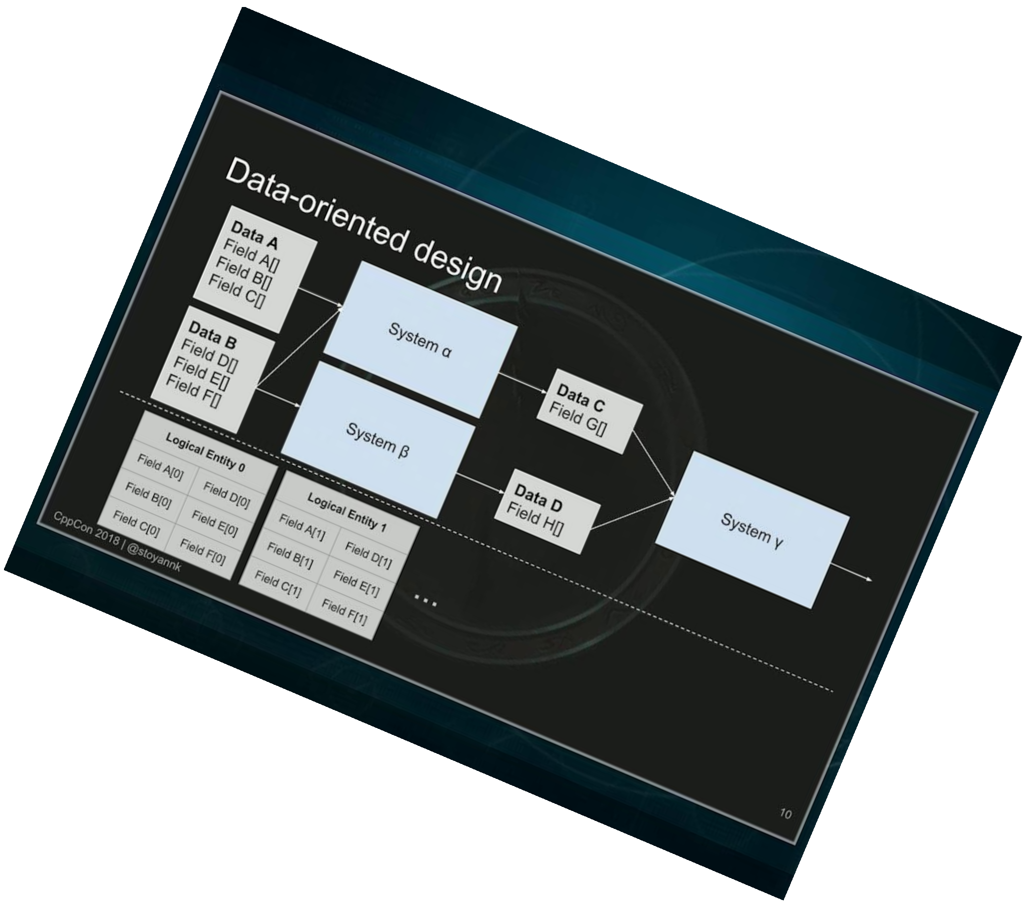
Alternatives to OOP

By Kimon Togrou





Contents

Introduction

First off, I chose to write this in English because of the technical terms that I'm going to be dealing with and the fact that everything I'll be studying on the subject is in English, also partly because I might want to post this online at some point.
I have chosen this subject because I feel that we get taught to think in a very specific way when it comes to developing software (Object-Oriented), and if I look on the web there is an increasing amount of prominent and very experienced software engineers that are speaking against this way of writing software, so I thought that it would be interesting to find out why that is. I want to figure out what we can learn from these people, and if what they're saying holds any water so to say. I have always been into philosophy and that extends to software development philosophies and as my teacher probably picked up from pretty much day one, performance is a big interest of mine.

Problem Definition

I am hoping to have answered all these questions by the end of these weeks:

- Is Object Oriented Programming slowing down our programs?
    - What is Data-Oriented-Design (DOD)?
    - What are the benefits of DOD vs OOP?
    - Which languages suit themselves best to DOD?
    - Can you write Data-Oriented in C#?
    - What is the difference between Array of Structs (AoS) and Struct of Arrays (SoA) and why is it relevant?
    - What kind of project/company should use which design method?
    - Does DOD compromise structure and maintainability for performance?

Method

As this is a very technical subject and there isn't a lot of readily available information on it, most of my work will consist of research although I am hoping to produce at least 1 or 2 code examples.

- I will watch all the talks I can find on the subject (which are about a handful)
- I'll be making a code example to test AoS vs SoA and see what runs the fastest
- Read forums and study the few practical examples I can find
- I would like to have a program developed in OOP and then make another version where I will try to apply DOD principles to look at any perceivable code and performance differences and code metrics

Planning[1]

I have four weeks to hand in this assignment, but I have spent first 3 days trying to come up with an angle for this subject and looking for sources so I will be planning the remaining days:

First Week:

- May 1st – 3rd: Watch talks and general reading/research fundamentals

Second Week:

- Write about the talks
- Research benefits of DOD and DOD within C# and other languages

Third Week:

- Write about which languages support DOD best
- Write about which kind of projects/companies fits DOD the best
- Research AoS vs SoA

Fourth Week

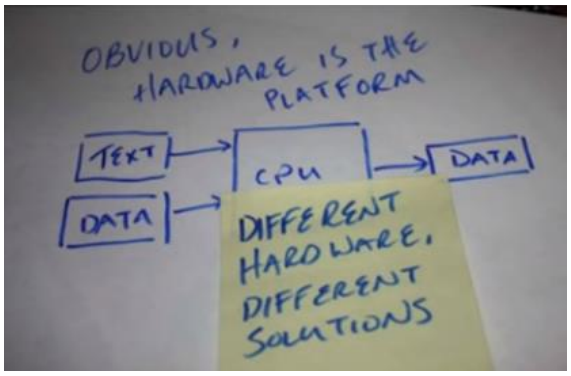
- Write a code example for AoS vs SoA
- Write a bigger code example if possible and compare metrics
- Write conclusion

[1] I couldn't figure out a productive way to set up a scrumboard as this is a very research heavy subject and felt it would be too vague.

Preface:

Let’s start by lightly defining Object-Oriented Programming (OOP) first so we have a point of comparison. OOP was established as a way of designing software from the idea of world modeling. It’s the presumption, that if we have different kinds objects like three types of “chairs” in our program there must be some generally defining characteristics, which we perceive, when we look at it in the real world, that makes a “chair” a “chair”, and therefore we should make a base class that hold the general info that all “chairs” share and all the individual objects should be derived from that base class. Basically, software/code is the platform, and it’s putting more importance on the fact, that we as people can find the “ideal” way to abstractly represent the data than on the fact that computers and more specifically CPU’s want the data laid out in a specific way. There is more to it than this of course, but this will do for now.

Mike Acton CppCon 2014 Talk


OBVIOUS, HARDWARE IS THE PLATFORM
TEXT
CPU
DATA
DATA
DIFFERENT HARDWARE, DIFFERENT SOLUTIONS

I thought it best to start by watching this talk, because as far as I can tell this talk is where the term Data-Oriented Design gained popularity and it framed this type of conversation for other people to talk about their own view of designing software in alternative ways to OOP.

It should be noted that Mike Acton is a game developer, and so he is very much coming at this from a perspective of squeezing as much out of the hardware as possible, but he also notes throughout the talk that his opinion wouldn’t change, if he was going to develop other types of programs.

Mike Acton’s point is that, since the purpose of all programs is to transform data from one form to another, the way we write programs should reflect the most efficient ways of doing this. It’s the presumption that because all we are doing is transforming data from one form to another, that if we don’t understand the cost of solving that specific problem, we don’t understand the problem. In other words; the hardware is the platform. We have a finite range of hardware constraints, and so we should make the most of it. From his point of view world modeling leads to huge programs that are so separated and abstract with unrelated data structures and transforms that tries to idealize the problem rather than solving the very specific data transformation that is required in the best/fastest way. He believes the programmers responsibility is for the data and not the code, the code is simply a tool to transform one set of data to another.

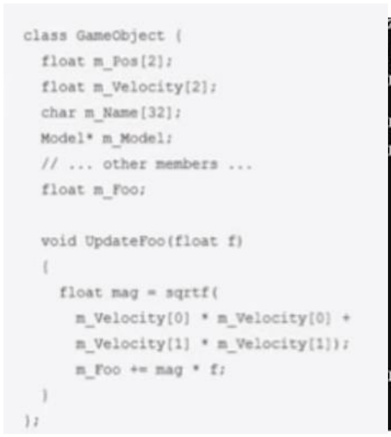
Make Acton believes that OOP perpetuate 3 lies:

1. Software is the platform
2. Code should be designed around a model of the world
3. Code is more important than data

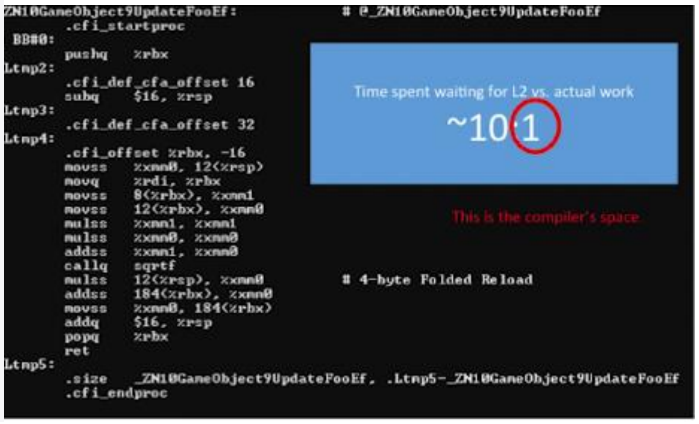
And that this philosophy leads to:

- Poor Performance
- Poor Stability
- Poor Testability
- Poor Concurrency
- Poor Optimization options

Another very interesting thing that he talks about is that compilers can’t reason well about the data in many situations. In a very simple example, he showcases that the compiler spent roughly 90% of the time missing the cache instead of doing actual work.



```
class GameObject {
  float m_Pos[2];
  float m_Velocity[2];
  char m_Name[32];
  Model* m_Model;
  // ... other members ...
  float m_Foo;

  void UpdateFoo(float f)
  {
    float mag = sqrtf(
      m_Velocity[0] * m_Velocity[0] +
      m_Velocity[1] * m_Velocity[1]);
      m_Foo += mag * f;
  }
};
```



```
ZN10GameObject9UpdateFooEf:              # @_ZN10GameObject9UpdateFooEf
        .cfi_startproc
 BB#0:
        pushq   %rbx
Ltmp2:
        .cfi_def_cfa_offset 16
        subq    $16, %rsp
Ltmp3:
        .cfi_def_cfa_offset 32
Ltmp4:
        .cfi_offset %rbx, -16
        movss   %xmm0, 12(%rsp)
        movq    %rdi, %rbx
        movss   8(%rbx), %xmm1
        movss   12(%rbx), %xmm0
        mulss   %xmm1, %xmm1
        mulss   %xmm0, %xmm0
        addss   %xmm1, %xmm0
        callq   sqrtf
        mulss   12(%rsp), %xmm0          # 4-byte Folded Reload
        addss   184(%rbx), %xmm0
        movss   %xmm0, 184(%rbx)
        addq    $16, %rsp
        popq    %rbx
        ret
Ltmp5:
        .size   _ZN10GameObject9UpdateFooEf, .Ltmp5-_ZN10GameObject9UpdateFooEf
        .cfi_endproc
```

Time spent waiting for L2 vs. actual work
~10:1
This is the compiler's space

Stoyan Nikolov CppCon 2018 talk

Stoyan Nikolov is also a game developer, which I think is largely going to be an ongoing pattern in my research as game developers out of absolute necessity must be worried about performance and deadlines.

His starting assertion is that OOP marries data with operations, and that this is not a very appropriate way to structure a program if you are concerned about:

- Performance
- Scalability
- Modifiability
- Testability

Nikolov says that you should start building your program with your most common operation in mind.

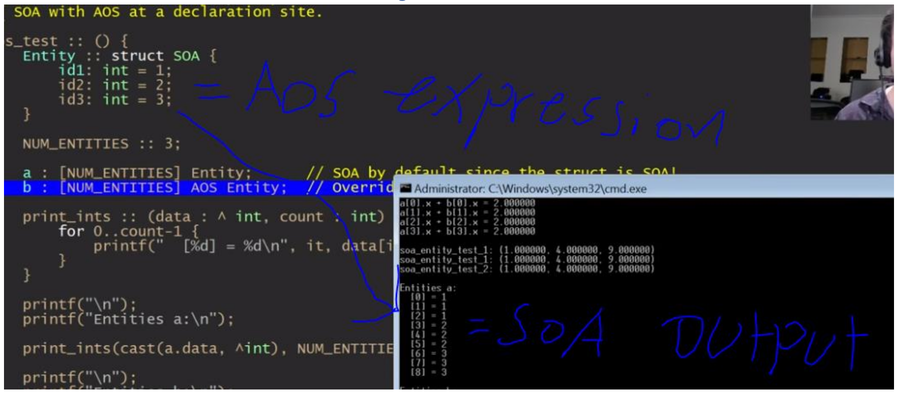
He has some much-appreciated practical guidelines, he lays out, that I have slightly modified for clarity.

- Separate data from logic
    - Structs and functions live independent lives
    - Data is regarded as information that must be transformed
- The logic embraces the data
    - Does not try to hide it
    - Leads to functions that work on arrays/linear data structures
- Recognizes data according to its usage
    - If we aren't going to use a piece of data, there is no reason to pack it together with the data we need to access a lot
- Avoid "hidden state"
    - Avoid making a lot of unnecessary states. Separate collections that are based on active/inactive states
- No virtual calls
    - There is no need for them (v-tables can slow things down because you'll be waiting for your function reference which translates to cache misses)

Nikolov’s conclusion:

- Multithreading in DOD is way easier since the data is separated more clearly, in that you separate object states into their own collections depending on the state
- Testing is easier since you don’t have to write a lot of mock code for the unit test because of multiple states and so on, all you have to do is test your functions, which are now simple input and out functions with very simple data
- It’s no longer as hard to modify your program, as you don’t have these base classes that are interconnected in very abstract and complicated ways.
- DOD is a tool and has it’s downsides like every other tool
    - Correct data separation is hard to accomplish
    - Quick data modifications can be hard because you must think about if the introduction of a new field/property is going to mess up your performance where in OOP you don’t care
    - It’s hard for newcomers to learn to think in this way when they are accustomed to OOP
    - Currently popular languages are not encouraging it
- What to keep from OOP?
    - Third party libraries
    - Some API systems might need it
    - It can be good for higher level systems
- Neither philosophy is a silver bullet
    - They both have their places

Jonathan Blow and Project “Jai”[2]



```
SOA with AOS at a declaration site.

s_test :: () {
  Entity :: struct SOA {
      id1: int = 1;
      id2: int = 2;
      id3: int = 3;
  }

  NUM_ENTITIES :: 3;

  a : [NUM_ENTITIES] Entity;       // SOA by default since the struct is SOA!
  b : [NUM_ENTITIES] AOS Entity;   // Overrid

  print_ints :: (data : ^ int, count : int)
      for 0..count-1 {
          printf("  [%d] = %d\n", it, data[i
      }
  }

  printf("\n");
  printf("Entities a:\n");

  print_ints(cast(a.data, ^int), NUM_ENTITIE

  printf("\n");
```

```
Administrator: C:\Windows\system32\cmd.exe
a[0].x + b[0].x = 2.000000
a[1].x + b[1].x = 2.000000
a[2].x + b[2].x = 2.000000
a[3].x + b[3].x = 2.000000

soa_entity_test_1: (1.000000, 4.000000, 9.000000)
soa_entity_test_1: (1.000000, 4.000000, 9.000000)
soa_entity_test_2: (1.000000, 4.000000, 9.000000)

Entities a:
  [0] = 1
  [1] = 1
  [2] = 1
  [3] = 2
  [4] = 2
  [5] = 2
  [6] = 3
  [7] = 3
  [8] = 3
```

Jonathan Blow is a game developer, and in the last couple of years, he has been making his own language, as he has grown increasingly tired of using C++.

He makes a lot of the same points as others I’ve mentioned so far, but he adds that many languages are made in the pursuit of high-level expressivity and abstraction to make it simpler to program as everything is well separated. Blow believes that this often creates a paradigm in which the program becomes way more complex as your project progresses than it has to be.

He describes his language as “a language for games”, but it is a general-purpose language.

- Primary goals of the language
  - Friction reduction
  - Joy of programming
  - Performance
  - Simplicity
  - Designed for good programmers
- Sub goals
  - Fast compilation
  - Very specific error messages
- What it shouldn’t be
  - Big agenda language
    - A language that is trying to apply one goal to every area of its scope
  - Focus on the fact that bad programmers exist
    - Make everything safe at the cost of freedom and performance
  - RAII (Resource Acquisition Is Initialization)
    - Constructors and destructors and so on causes friction

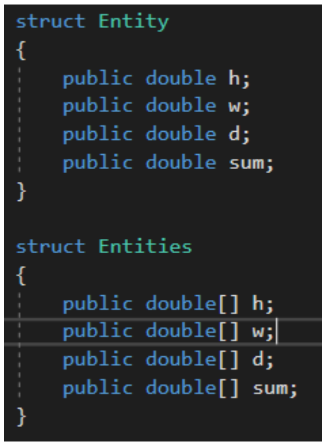
[2] This is based on a series of videos from his YouTubes channel in which he describes his ideas and demos his new language – references at the end of the document

After looking at some of his demos, I have seen some very interestingly executed ideas. He made it so that you can you do a lot with memory access whilst retaining a lot of high-level expressivity. The most exciting thing for me so far is his use of “AoS/SoA” and “using”. The language supports you writing a struct as AoS but using “SoA” as a keyword for the compiler then makes the memory access pattern SoA which he says improves performance in most areas without compromising the pattern, that we are very used to in OOP. That can be done on the fly too. You can have an AoS setup until the point you need to do something with it, like looping through an array of structs if need be.

He uses the “using” keyword in multiple ways, for one, he has made it so that you at any time/scope of your program can use it to pull members into your namespace to shorten how much you have to write to access certain properties or functions within a given context. He also uses the keyword as a way to do polymorphism in a less expensive way, so basically, what you can do is, use it as an argument in a function, and then you can put anything that is “using” that struct into your function as an argument. It retains a lot of the core features of inheritance without actual inheritance, so the benefit would be that you get to avoid v-tables which cause a lot of cache misses.

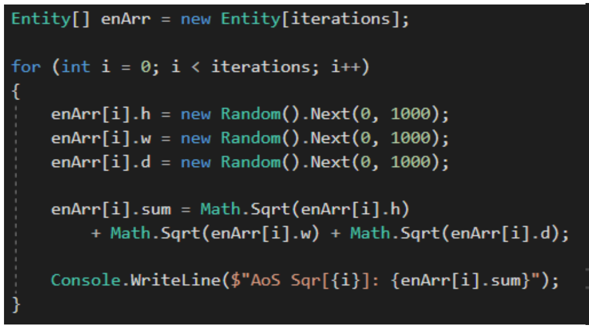
AoS vs SoA

When we are taught to write OOP, we usually start by creating an object, maybe a person. This object has some attributes and so we type in the properties, a name maybe, perhaps an age. After this it's quite natural to go on to creating a list of these objects, and then loop trough them to look at average age or some other calculation. This usually leads us to structure most of our code in Array of Structs/Objects. The adherents to DOD talk a lot about the Structure of Arrays pattern.

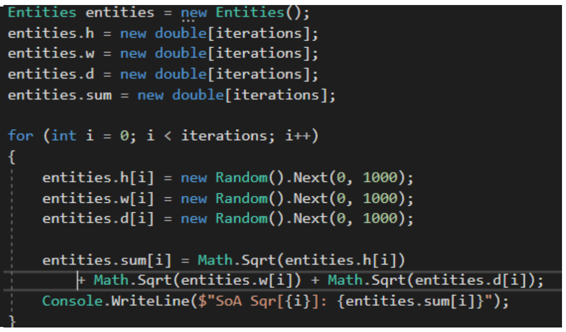


```
struct Entity
{
    public double h;
    public double w;
    public double d;
    public double sum;
}

struct Entities
{
    public double[] h;
    public double[] w;
    public double[] d;
    public double[] sum;
}
```

I tried to come up with a simple example in C# to try to understand what the difference is. I started by declaring two different structs of course, one in AoS and another in SoA. The Idea was to loop through these two 10000 times doing the same mathematical operation and output the result, and then use a stopwatch to see how they compare.

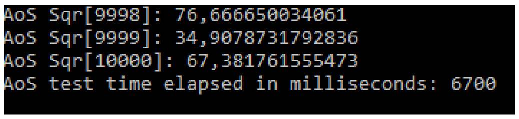


```
Entity[] enArr = new Entity[iterations];

for (int i = 0; i < iterations; i++)
{
    enArr[i].h = new Random().Next(0, 1000);
    enArr[i].w = new Random().Next(0, 1000);
    enArr[i].d = new Random().Next(0, 1000);

    enArr[i].sum = Math.Sqrt(enArr[i].h)
        + Math.Sqrt(enArr[i].w) + Math.Sqrt(enArr[i].d);

    Console.WriteLine($"AoS Sqr[{i}]: {enArr[i].sum}");
}
```

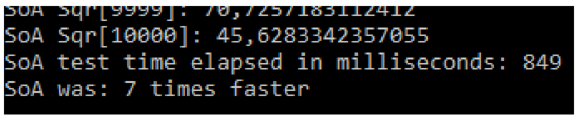


```
Entities entities = new Entities();
entities.h = new double[iterations];
entities.w = new double[iterations];
entities.d = new double[iterations];
entities.sum = new double[iterations];

for (int i = 0; i < iterations; i++)
{
    entities.h[i] = new Random().Next(0, 1000);
    entities.w[i] = new Random().Next(0, 1000);
    entities.d[i] = new Random().Next(0, 1000);

    entities.sum[i] = Math.Sqrt(entities.h[i])
        + Math.Sqrt(entities.w[i]) + Math.Sqrt(entities.d[i]);
    Console.WriteLine($"SoA Sqr[{i}]: {entities.sum[i]}");
}
```

I'm taking three variables and generating a random number and then calculating the square root of each, adding them together and putting it into the sum variable and output that. Here are the results:



```
AoS Sqr[9998]: 76,666650034061
AoS Sqr[9999]: 34,9078731792836
AoS Sqr[10000]: 67,381761555473
AoS test time elapsed in milliseconds: 6700
```



```
SoA Sqr[9999]: 70,7257183112412
SoA Sqr[10000]: 45,6283342357055
SoA test time elapsed in milliseconds: 849
SoA was: 7 times faster
```

Further Experimentation:

I then started to try with different amounts of iterations and found that there is a huge difference in the performance gain. When I tried with 100/1000 iterations it was more like 3-4x faster with SoA. My initial thoughts were that I was accessing different caches(L1, L2, L3) depending on the initialization size of my variables, which would make sense, because the bottleneck I was dealing with was not the mathematical operation, but memory access speed. Thus, the more time consuming it is to access the data the more benefit we will get from SoA, as it leads to less or zero cache misses. This means we don't have to travel over to the given cache to access the data more times than we absolutely need to. My expectation after seeing the result was that the difference should get larger the more data we need to access, because the more data there is to loop through the further back in the hierarchy of memory-based hardware we must go. The fastest (L1 cache) has the least amount of memory, whereas the slowest (Ram) has the largest amount and in between we have L2, L3. So, if we have to go to the L3 cache a lot then SoA is said to have a bigger impact than if we only needed to access L1.

If I started the program multiple times in quick succession with smaller amounts of iterations, then the performance boost seems to diminish. The same thing happens if I put in any number over 10000 iterations. If I put in 20000 it seems that after the first 10000 iterations, the speed of looping through the remaining data seems exponentially faster. It looks as if the compiler is finding a pattern or maybe there is something in memory that helps.

It turns out that the time it takes to do 10000 calculations is about the time it takes for the compiler or the runtime processes to get going, so I found out that if I run an AoS test before running the timed one, the performance boost seems trivial, this makes me think that there is something wrong with the way I've setup the experiment which makes the calculation the bottleneck.

Googling a lot and reading forum posts about SoA in C# lead me to the conclusion that the way Garbage Collecting works in C# is somehow influencing the experiment and because of the closed nature of Garbage Collection and other run time processes behind C# there is really no reliable way to know what's going on under the hood.

Visual Studio Features

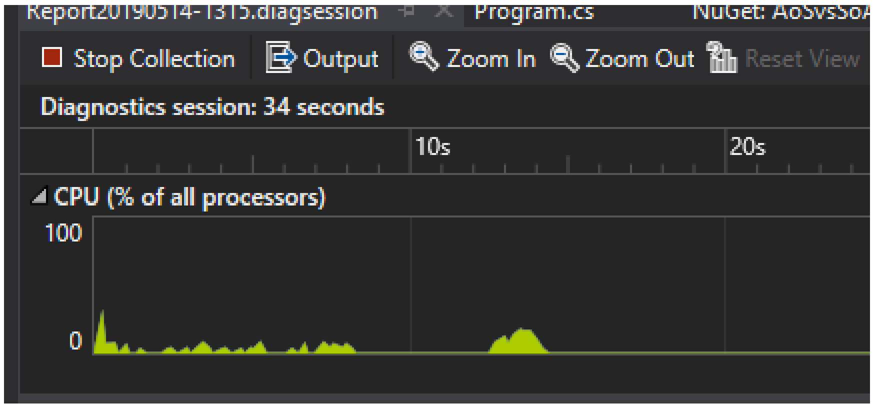
Code Metrics:

I tried to get code metrics to work on .Net Core applications through plugins to get ready for my bigger experiment, but ultimately it was too much hassle and so I just created a separate .Net Framework project that I can copy paste files into and view the metrics from there.

Compiler:

This was partly an effort to see if I could diagnose the AoS vs SoA example.

Trying to figure out the best way of viewing memory access and CPU cache cycles in Visual Studio was a very time-consuming and ultimately fruitless undertaking, it seems as though people really aren't concerned with what is happening behind the hood when running their code and so googling left me with very little to go on. The easiest feature to find was the performance profiler:


Report20190514-1315.diagsession
Program.cs
NuGet: AoSvsSoA
Stop Collection
Output
Zoom In
Zoom Out
Reset View
Diagnostics session: 34 seconds
10s
20s
CPU (% of all processors)
100
0

This doesn't really tell me at lot that I can't just output through my code. It shows me some graphs, but what I was looking for was more of a text-based representation of what the compiler is doing, what/how is data being stored in memory. How many cycles did it take to read something? Basically, some information that can help me reason about the data, to see if I'm doing something expensive, and if there might be better ways of structuring functions. Just knowing that a function or line of code is heavy doesn't tell me much about how to change it for the better.

Conclusion

What is Data-Oriented Design?

Data-Oriented Design is the idea that data should be the driving force on how you structure your program. In a lot of OOP books, you will read about the End User which refers to the programmer who needs to use the code someone is writing thus treating software as the platform. In DOD the End User is thought of as the person who is going to be using your program I.E the consumer. The idea is that you should write performant code to give the best experience for the End User, therefore the hardware must be thought of as the platform. DOD focuses on the fact that there is a range of hardware limitations, and we should write our program in the most efficient way as possible to capitalize on the finite amount of resources, and not concern ourselves with our abstract idea of world modeling.

Guidelines

- No inheritance and virtual calls
- Use simple data
- When in doubt use linear arrays
- Simple input and output functions
- If there are many, SoA is usually better
- Limit the amount of states in the program (bool usage)
- Separate data and functions/methods
- Separate arrays based on states

Benefits of DOD:

- It's all about the data, which means you know you're dealing with the actual problem of transforming data efficiently
- Cache utilization, DOD makes you lay out the data in a contiguous and homogeneous way which is what the CPU ultimately wants
- Parallelization becomes much easier because you localize the problems with small functions that have input and output data which is easy to get running on multiple threads
- Testing becomes a lot simpler too for the same reason. When you are largely dealing with low level data transformations like small functions that have an input and an output, it's easy to check if the data transformation went as it should without having to create a lot of mock code to simulate all of the dependencies and state heavy scenarios that OOP encourages

- Modularity is a big bonus as there aren't as many dependency problems, like classes that are interconnected in very complicated and abstract ways, this makes it so you can move pieces around more freely if you need to experiment

Cons:

- There are not really any good sources on the subject and that makes it hard to learn
- It's very difficult to "unlearn" the OOP way of thinking about software although I am not sure how much harder if at all it would be to learn OOD if you are a beginner
- Popular high-level languages are not encouraging DOD
- It's said that you lose a little on the maintainability metric

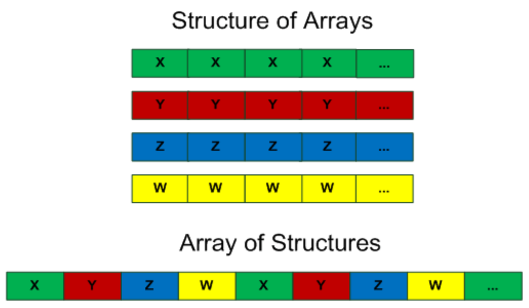
Which languages are most suited to DOD?

From the research that I've gathered so far there is only one language besides languages like C (low level languages) that is ideal for this approach. Jonathan Blow's new language is the only one, that from the start has been built and designed to adhere to the Data-Oriented principles. Apart from Jai, I would say that C++ is another candidate, even though it like C# is also very firmly placed in the Object-Oriented world, at the very least doesn't have Garbage Collection and it's easy to look at assembly and write out specific code to the CPU.

Can you write Data-Oriented in C#?

My thought out the gate was that C# being a Garbage Collected language and mostly focused on OOP means that you can't write Data-Oriented, but what I quickly found out is that, DOD is not really about rigid frameworks or patterns, but it's that you put the data first, and there are simple ways that you can do this even in C#. That being said; it's obvious to anyone that investigates this, that C# is not the best choice if your primary goal is to run fast. If to accomplish that you need full control of memory allocation and such, my personal conclusion is that you should look elsewhere.

What is the difference between Array of Structs (AoS) and Struct of Arrays (SoA) and why is it relevant?



The core difference between these two patterns is supposed to be the way they are stored in memory. In AoS it lays out the variables in the way you might imagine objects next to eachother would look like. If you have 4 variables per object it lays out each set of those 4 variables one after the other. If we only needed to do operations with two of the variables, then in AoS we would go through all of this unneccesay data in each set as seen in the picture. Furthermore it's heavier on the CPU to run through the data when it's not next to eachother because of the way CPU's are structured. Where AoS should be useful is if you need to do operations with all the variables and if you do it one at a time. With SoA data gets layed out contiguously as a set of each data type. This for one means that we don't loop through data that we don't need. There are also very specific CPU bonuses associated with SoA:

- Using the whole cache line
    - As explained above
- Prefetching
    - Fetching data from slower memory and pulling it into faster memory before it is needed
- SIMD (Single instruction, multiple data)
    - Parallelization on one core

In C# this seems to be trivial, but in C++ it is said to have huge effects on performance.

What kind of project/company should use which design method?

It's obvious just by the adherents of DOD, that game development is where this design method is going to be most useful, as you need to get the most out of the machine to create an enjoyable experience for the player. You could argue that API's and such benefits a lot from OOP as it's an environment where interfaces and such are useful to create templates for the data going to and from the API.

Really what it comes down to though, is what you want to focus on. OOP can create a lot of friction if you are making a program which you know needs performance optimization. If you know most of the program is going to hinge on multithreading and parallelization you might want to reconsider your use of OOP.

General Conclusion

OOP is slowing down the programs we are writing, as soon as you introduce inheritance and virtual functions you are bound to create cache misses. I must admit I've become biased toward DOD. I think it comes down to the fact that there are a lot of resources being spent inefficiently in the OOP world. I also think it's counterintuitive to think of the End User as the programmer who is going to be using our code. What I ultimately want from my programs is the best experience for the consumer, and if the culture would one day shift to DOD, I think that many people would also find it easier in the long run. My problem with OOP is that you end up with a very complex representation of simple transformations. Yes, in isolation the code snippets are easy to understand, but figuring out the larger context can be very difficult. The same thing is true when you are writing the program. At first it makes sense, because you're only dealing with a handful of classes, but the program quickly becomes very large, complicated and suddenly you will find yourself breaking the world modeling patterns just to get something done in a timely and efficient manner. Testing becomes harder because there are many dependencies. Changing and/or more specifically if you want to move code around, you must change many references that you just broke, which makes experimentation a nightmare. What I like about OOP is the fact that you can write libraries that help you write working code very quickly.

Reflection

If I look back on my work, I wish I hadn't spent as much time on the AoS vs SoA example and wish I had gone on to the last question about maintenance. I just felt like it was important as it is one of the main performance boosts that is talked about and is the easiest thing to relate to CPU cache usage. I also spent a lot of my time trying to see if there was a way to look under the hood of C# and should have realized way sooner than I did, that it would be a dead end because of Garbage Collection.

Exam Preparation

- Hopefully show AoS vs SoA in a C++ example
- I would like to have an example ready of a project rewritten from OOP to DOD and see if I can answer my last question definitively

References

Mike Acton Talk:
https://www.youtube.com/watch?v=rX0ItVEVjHc

https://www.youtube.com/watch?v=p65Yt20pw0g

Stoyan Nikolov Talk:
https://www.youtube.com/watch?v=yy8jQgmhbAU

Jonathan Blow Talks:
https://www.youtube.com/watch?v=TH9VCN6UkyQ&list=PLmV5I2fxaiCKfxMBrNsU1kgKJXD3PkyxO

What Is CPU Cache? Article:
https://www.makeuseof.com/tag/what-is-cpu-cache/

Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP):
http://gamesfromwithin.com/data-oriented-design

Forum Posts:
https://stackoverflow.com/questions/29192679/memory-layout-optimization-in-c-sharp

https://stackoverflow.com/questions/56255090/does-c-sharp-allocate-array-of-structs-as-structure-of-arrays-in-memory

AoS vs SoA:
https://www.youtube.com/watch?v=ScvpoiTbMKc

https://www.youtube.com/watch?v=Dj3lkfIEUyA&t