Unity at GDC - C# to Machine Code

Current options for delivering C#

- Mono
    - Primary focus: *Quick iteration and sandboxing, enable productivity*
    - Code quality: Not awesome
- IL2CPP
    - Primary focus: *Shipping games on non-JIT platforms*
    - Code quality: Mixed results- some ok, some bad
    - Difficult to iterate on code quality due to additional C++ step

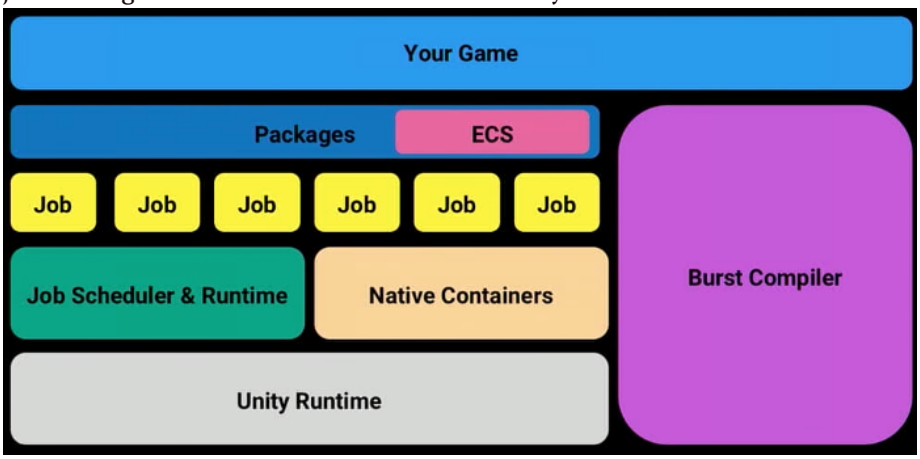
C# to Machine Code

Burst Complier(2018.2 release)

- Custom in-house compiler stack to take advantage of *better defaults*
- Primary focus: *Performance*
- A performance sandbox

The Big Picture

Taking a very deep integration approach here that the burst compiler is a package. And it's deeply integrated with our jobs and the way we type them up. It's also integrated with the native containers that it has awareness of these things. Aand it cooperates with the unity runtime to deliver a completely deep integrated compiler stack that it's not like we're taking some executable and just bolting it onto the side like no this is a unity effort.



High-performance c#

HPC#是一个C#的完美子集(a perfect subset of c#)，满足一些严格的约束(tough constraints)：

- Constraints:
    - No class types
    - No boxing
    - No GC allocation
    - No exceptions for control flow

这些约束意味着:

- It's a highly optimizable language
- It's more feasible to statically analyze for safety issues

```
[ComputeJobOptimization]
struct RemoveDeadJob : IJob
{
    public bool playerDead;
    [ReadOnly] public EntityArray Entity;
    [ReadOnly] public ComponentDataArray<Health> Health;
    public EntityCommandBuffer Commands;

    public void Execute()
    {
        for (int i = 0; i < Entity.Length; ++i)
        {
            if (Health[i].Value <= 0.0f || playerDead)
            {
                Commands.DestroyEntity(Entity[i]);
            }
        }
    }
}
```

It's still C# 它仍然是C#

- All basic types
- Structs
- Enums
- Generics
- Properties
- Safe sandobx
    - Even for parallel code

Fixing C# ?

"Just fix general purpose C#" is an intractable problem. 直接修复通用目的的C#是一个很难的问题。

- Garbage collection
- Managed objects = cache misses
- Boxing
- OO legacy code

Instead, let's focus on the subset of C# that we can highly optimize. 相反，我们只关注C#的子集 (可以高度优化那部分)。

How Burst Works

- We compile assemblies normally using the C# compiler
- Burst asynchronously recompiles job kernels in the editor
    - Consumes .Net IL (currently opt-in per job)
    - Transforms into LLVM IR, adding our high-level metadata
    - Runs aggressive LLVM optimizations, JITs
- Ahead of time workflow will integrate with IL2CPP (work ongoing)
    - Precompile all job kernels off line

Burst Inspector

在菜单栏: **Jobs > Burst > Open inspector ...** 打开

Burst + Unity = Better Together

- Context-aware alias analysis (partially implemented today)
- Precision and determinism controls (planned)
- Higher-level SOA/AOS data transforms (research project)

Aliasing

"In computing, aliasing describes a situation in which a data location in memory can be accessed through different symbolic names in the program.
...
As a result, aliasing makes it particularly difficult to understand, analyze and optimize programs" - Wikipedia

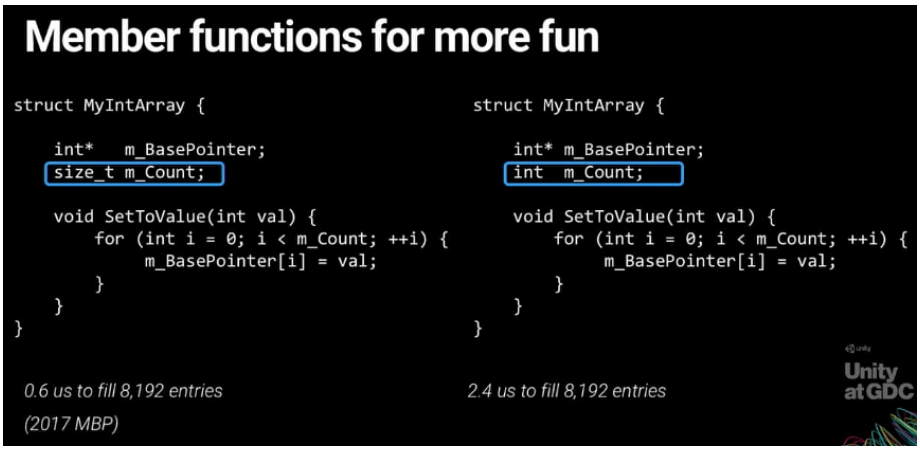
```
1  void foo(const float* a, const float* b, float*
   __restrict out)
2  {
3      for (int i = 0; i < 8; ++i)
4      {
5          out[i] = a[i] + b[i];
6      }
7  }
```

Great, `__restrict` solves all your problems?

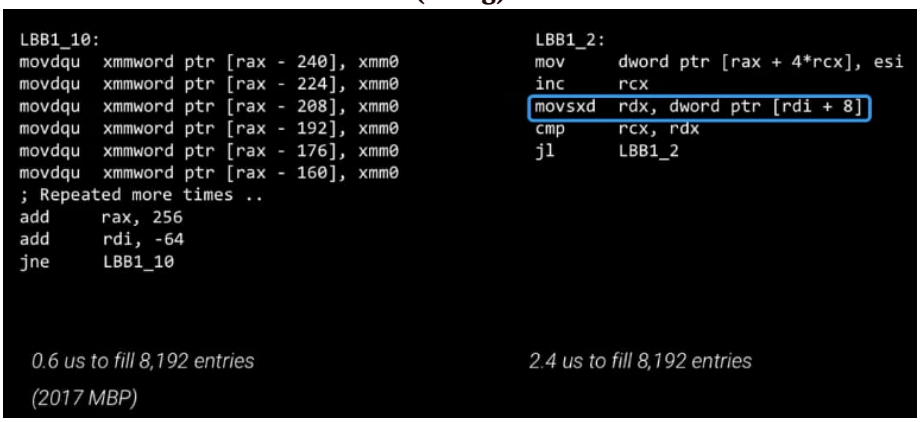
- No
- It's not safe to randomly sprinkle in `__restrict` in code
    - The keyword applies to a function parameter...
    - But the ramifications extend to all *call sites*, present and future

        - There are virtually no tools to tell you that you're doing it wrong on the call site
    - Also not useful for hidden pointers
        - Consider the pointer inside an std::vector or `List<T>` for example

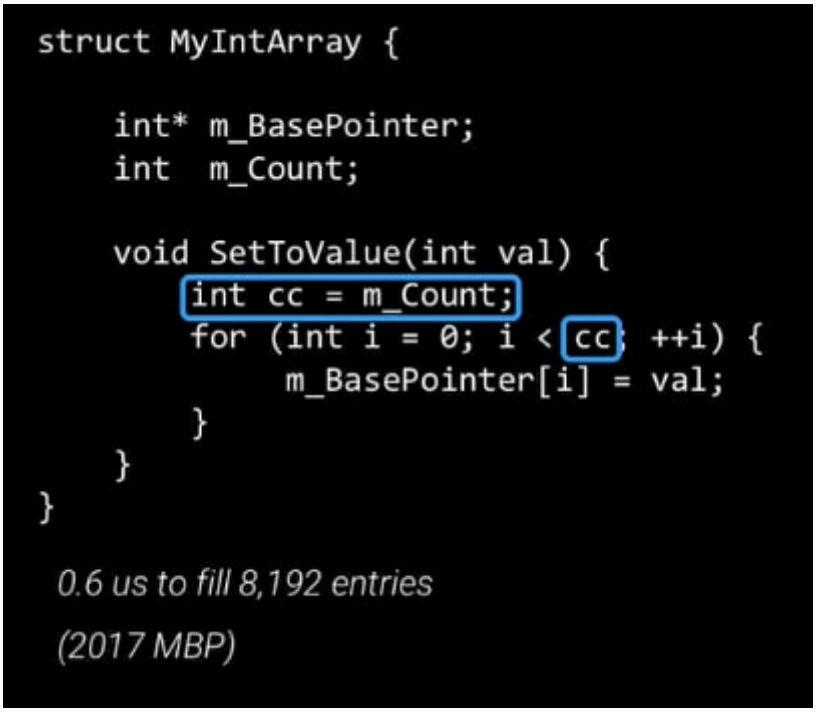
Memeber functions for more fun



Memeber functions for more fun (Clang)



One Fix

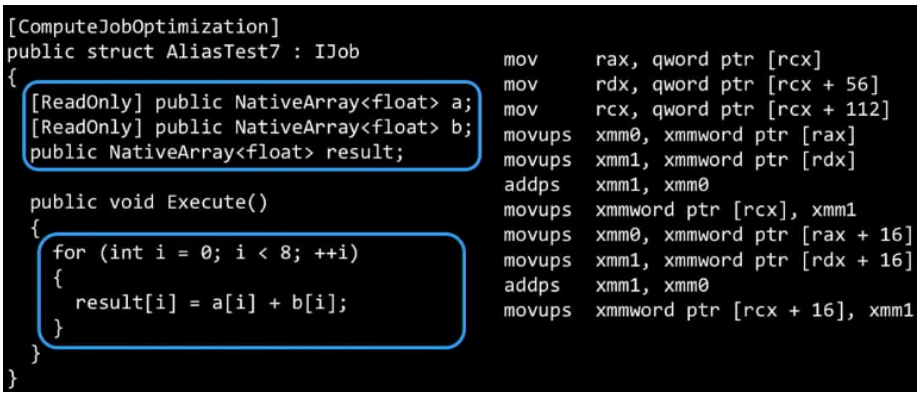


Aliasing Problem Summary

- Practically, aliasing typically prevents the use of SIMD instructions
    - Can easily measure 4x performance loss in many simple cases
    - SIMD leverages more HW resources and can greatly speed up a lot of workflows
- Additonally, causes reloads from memory (long slow dependency chains)
- Lack of contex means generic compilers can't solve this problem
    - Compilers have to be defensive and follow the rules
    - Need `__restrict` to be carefully placed in the right places to allow compiler to break the rules

Context-Aware Alias Analysis

- Our solution is to be context aware to automatically provide aliasing info
    - Our compiler runs on each job individually
    - We prevent jobs from getting scheduled with native buffers aliasing each other
    - Therefore, any pointers derived from unique native buffers are always alias free
- Not a perfect solution yet, but it gives much better default loop behavior
    - Enables auto-vectorization in more loops
    - Best of all, costs nothing for the programmer to maintain



Precision Control (planned)

- How much precision do you need for a particular section of code?
    - Answer is context dependent
    - Running with lower precision can give you nice perf wins
- Visual effects in the far distance?
- That one game-defining effect you're zooming into?
- Terrain path finding code?
- Your spell cooldown code?

Problems in current general-purpose compiler ecosystem

- Lack of control is the largest issue we see
- Working with half precision data is clumsy
    - Needs intrinsics and per-platform code
- -ffast-math is a ham-fisted way to treat a whole object file
- Hard to make sure compilers are using appropriate tricks
    - Reciprocal multiplication
    - Fast square roots

Determinism is an open problem space

- Critical for certain networking algorithms
    - E.g. RTS-style input networking needs deterministic simulation
- The important part is bitwise identical results on all platforms, not the precision
- Making simulation deterministic with FP math across different platforms = hard
    - No tooling
    - Fixed point integer path is deterministic but a big leap to rewrite all math code
- It seems no one is interested in solving this problem

Where we want to go

- Selectable precision settings per job
- Selectable determinism settings per job
- Both exposed through `[ComputeJobOptimization]` attribute
- These are orthogonal settings

What we have to figure out for determinism to work well

- Math library determinism
    - cos, sin and friends
- Comprehensive testing across CPUs and devices
- What will the performance hit be?
    - What fun IEEE out of spec behavior will bite us?

SIMD memory layout (research)

- Basically all hardware we target has SIMD hardware
- Using SIMD well is key to boosting CPU performance
- Layout of data is critical to enabling the use of SIMD instructions

Understanding SIMD

- Consider a hypothetical CPU computing A = B + C (with floats)

```
Register0 = Load B from memory
Register1 = Load C from memory
Register1 = Register0 + Register1
Store Register1 to memory at address of A
```

- SIMD makes the registers wider and does "more than one at a time"

```
Register0 = Load B0/B1/B2/B3 from memory
Register1 = Load C0/C1/C2/C3 from memory
Register1 = Register0 + Register1
Store Register1 to memory at address of A0/A1/A2/A3
```

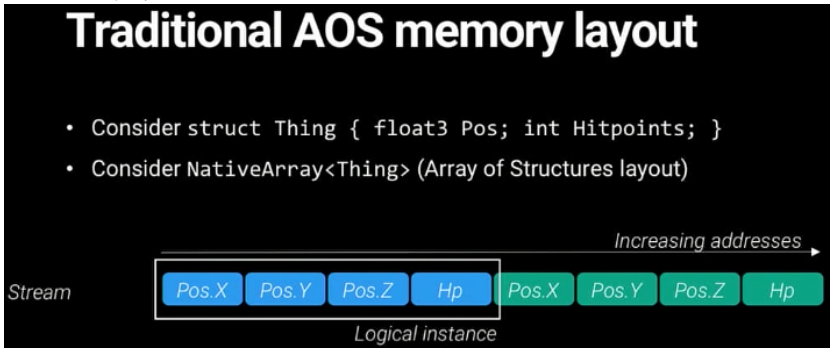
SIMD Constraints

- Consider this step: Load B0/B1/B2/B3 from memory
- To support this efficiently, we need to *already have* data in memory in that order
- Note that basic arrays of say ints/floats already work for this
    - As long as there are no intra-element dependencies
- The confusion comes from user-defined structs

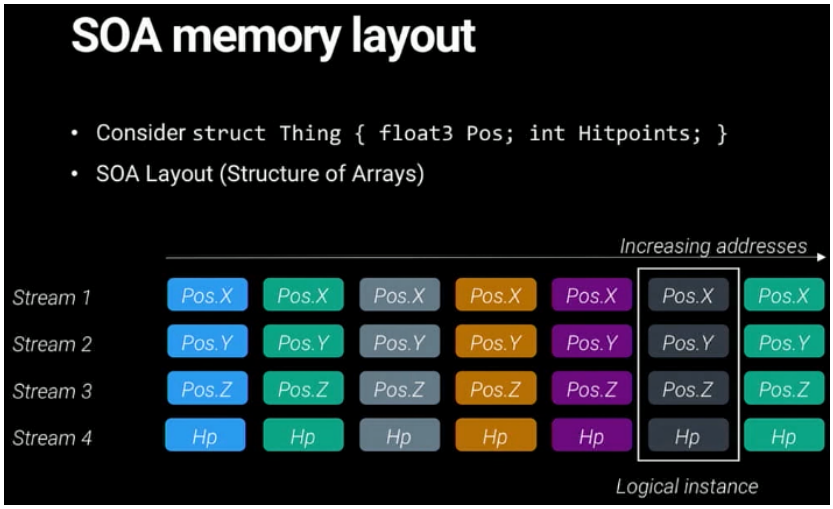
User Structs

- We tend to think of structs/classes as something with identity
    - That identity is typically a pointer/reference to the first byte
    - Pascal/C/C++/C#/ADA/...
- We're trained to abstract this way
    - Group together related attributes to make an "instance"
- But by grouping things together like that we also define memory layout

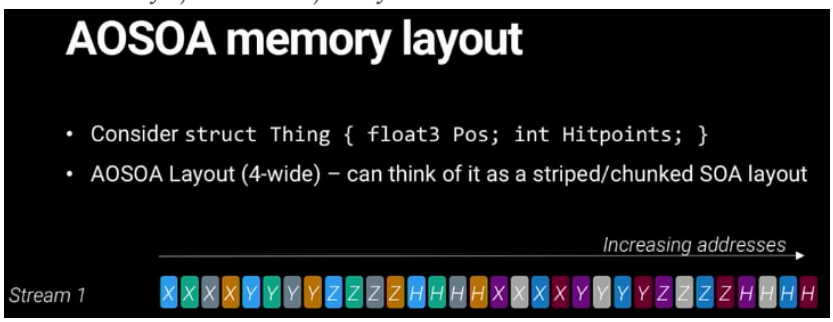
Traditional AOS memory layout

> *AOS: array of structure*
>
> 
>

SOA memory layout

> *SOA: structure of array*
>
> 
>

AOSOA memory layout

> *AOSOA: array of structure of array*
>
> 
>

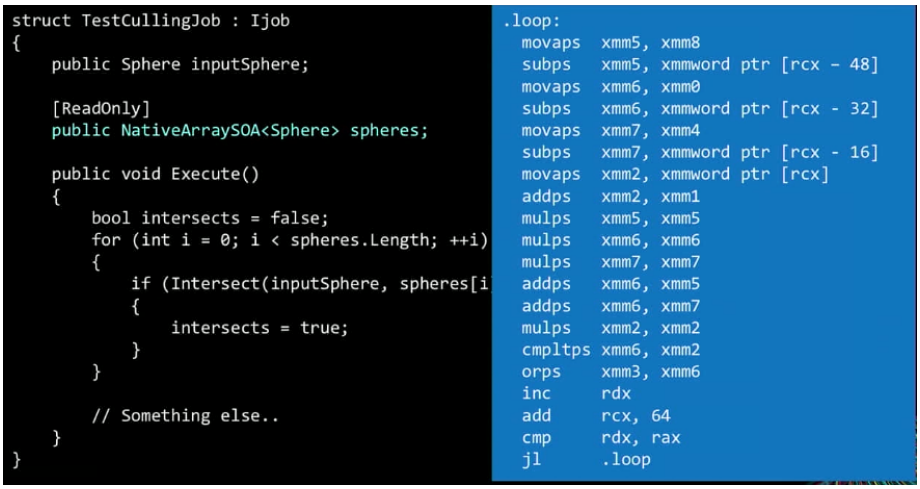
Where we want to go

- Add SOA layout aware containers
    - `NativeArraySOA<T>` - user default chunk width for platform SIMD register
    - Re-assemble user structs on read from C#
    - Split apart and scatter of write from C#
- In Burst, take advantage of knowledge of strided layout
    - Supporting the auto-vectorization of loops without swizzling
    - Feedback loop through Burst inspector when looking at code gen

```
// using NativeArray
struct TestCullingJob : IJob
{
    public Sphere inputSphere;

    [ReadOnly]
    public NativeArray<Sphere> spheres;

```

```
    public void Execute()
    {
        bool intersects = false;
        for (int i = 0; i < spheres.Length; ++i)
        {
            if (Intersect(inputSphere, spheres[i]))
            {
                intersects = true;
            }
        }
        // Something else...
    }
}

// Sphere
struct Sphere
{
    public float x;
    public float y;
    public float z;
    public float r;
}

// Intersect
bool Intersect(Sphere a, Sphere b)
{
    float dx = a.x - b.x;
    float dy = a.y - b.y;
    float dz = a.z - b.z;
    float dr = a.r + b.r;

    dx *= dx;
    dy *= dy;
    dz *= dz;
    dr *= dr;

    return dx + dy + dz < dr;
}

// using NativeArraySOA
struct TestCullingJob : IJob
{
    public Sphere inputSphere;

    [ReadOnly]
    public NativeArray<Sphere> spheres;

    public void Execute()
    {
        bool intersects = false;
        for (int i = 0; i < spheres.Length; ++i)
        {
```

```
61              if (Intersect(inputSphere, spheres[i]))
62              {
63                  intersects = true;
64              }
65          }
66          // Something else...
67      }
68  }
```



Conclusion

- We're trying to make a smarter compiler by teaching it more about Unity
- We're trying to break some new ground with Burst
    - We're only compiling Unity game code
    - Not making a general purpose language or tool
- HPC# is an optimizable C# subset that enables this

参考:

1. Unity at GDC - C# to Machine Code