東京工業大学
Tokyo Institute of Technology

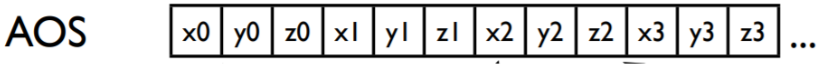
A C++/CUDA DSL for Object-oriented Programming with Structure-of-Arrays Layout

Matthias Springer
Tokyo Institute of Technology

CGO 2018, ACM Student Research Competition

AOS vs. SOA

- AOS: Array of Structures



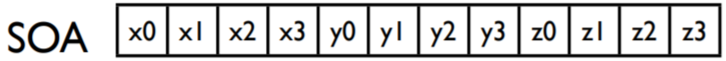
AOS													
	x0	y0	z0	x1	y1	z1	x2	y2	z2	x3	y3	z3	...

```
struct Body {
  float pos_x, pos_y, vel_x, vel_y;

  void move(float dt) {
    pos_x += vel_x * dt;
    pos_y += vel_y * dt;
  }
};

Body bodies[128];
```

- SOA: Structure of Arrays



SOA												
	x0	x1	x2	x3	y0	y1	y2	y3	z0	z1	z2	z3

```
float pos_x[128], pos_y[128], vel_x[128], vel_y[128];

void move(int id, float dt) {
  pos_x[id] += vel_x[id] * dt;
  pos_y[id] += vel_y[id] * dt;
}
```

SOA: Good for caching, vectorization, parallelization

AOS vs. SOA

- AOS: Array of Structures

```
struct Body {
  float pos_x, pos_y, vel_x, vel_y;

  void move(float dt) {
    pos_x += vel_x * dt;
    pos_y += vel_y * dt;
  }
};

Body bodies[128];
```

AOS

x0	y0	z0	x1	y1	z1	x2	y2	z2	x3	y3	z3	...

- SOA: Structure of Arrays

```
float pos_x[128], pos_y[128], vel_x[128], vel_y[128];

void move(int id, float dt) {
  pos_x[id] += vel
  pos_y[id] +=
}
```

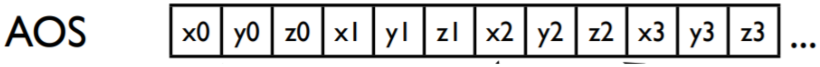
SOA

x0	x1	x2	x3	y0	y1	y2	y3	z0	z1	z2	z3

IDs instead of pointers

AOS vs. SOA

- AOS: Array of Structures

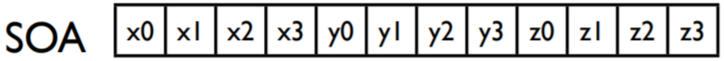


```
struct Body {
  float pos_x, pos_y, vel_x, vel_y;

  void move(float dt) {
    pos_x += vel_x * dt;
    pos_y += vel_y * dt;
  }
};

Body bodies[128];
```

- SOA: Structure of Arrays



```
float pos_x[128], pos_y[128], vel_x[128], vel_y[128];

void move(int id, float dt) {
  pos_x[id] += vel_x[id] * dt;
  pos_y[id] += vel_y[id] * dt;
}
```

- IDs instead of pointers
- No member of obj./ptr. operator
- No constructors, new keyword
- No inheritance
- No virtual function calls

Embedded C++ DSL

```
class Body : public SOA<Body> {
  public: INITIALIZE_CLASS
    float_ pos_x = 0.0;
    float_ pos_y = 0.0;
    float_ vel_x = 1.0;
    float_ vel_y = 1.0;

    Body(float x, float y) : pos_x(x), pos_y(y) {}

    void move(float dt) {
        pos_x = pos_x + vel_x * dt;
        pos_y = pos_y + vel_y * dt;
    }
};

HOST_STORAGE(Body, 128);
```

Use this class like any other C++ class:

```
void create_and_move() {
    Body* b = new Body(1.0, 2.0);
    b->move(0.5);
    assert(b->pos_x == 1.5);
}
```

Embedded C++ DSL

```
class Body : public SOA<Body> {
  public: INITIALIZE_CLASS
    float_ pos_x = 0.0;
    float_ pos_y = 0.0;
    float_ vel_x = 1.0;
    float_ vel_y = 1.0;

    Body(float x, float y) : pos_x(x), pos_y(y) {}

    void move(float dt) {
        pos_x = pos_x + vel_x * dt;
        pos_y = pos_y + vel_y * dt;
    }
};

HOST_STORAGE(Body, 128);
```

“Parallel” API (CPU+GPU):

```
Body* q = Body::make(10, 1.0, 2.0);

forall(&Body::make, q, 10, 0.5);
forall(&Body::make, 0.5);
```

Implementation Outline

```
class Body : public SOA<Body> {
  public: INITIALIZE_CLASS
    float_ pos_x = 0.0;
    float_ pos_y = 0.0;
    float_ vel_x = 1.0;
    float_ vel_y = 1.0;

    Body(float x, float y) : pos_x(x), pos_y(y) {}

    void move(float dt) {
        pos_x = pos_x + vel_x * dt;
        pos_y = pos_y + vel_y * dt;
    }
};

HOST_STORAGE(Body, 128);
```

During assignment of float, conversion to float

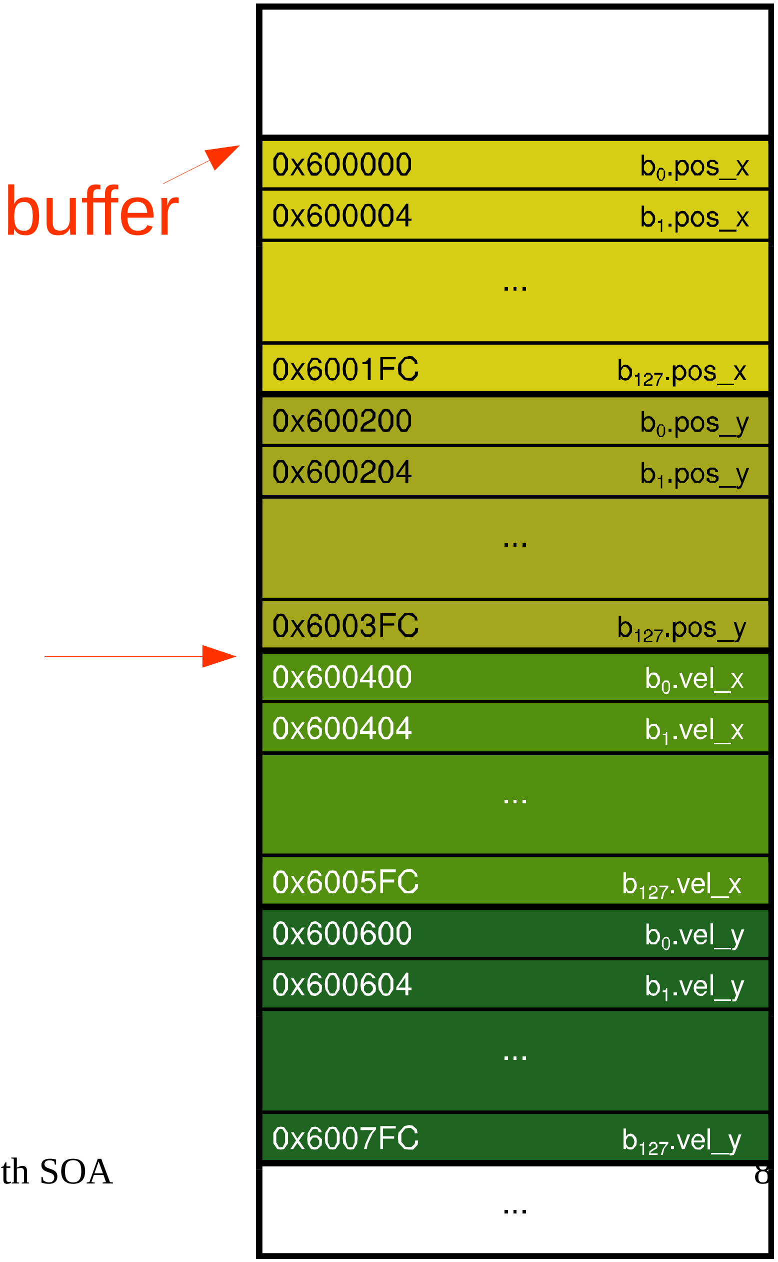
Calculate physical memory location inside `buffer`

```
char buffer[128 * 16];
```

Implementation Outline

e.g.: `float x = b127->vel_x;`

beginning of array

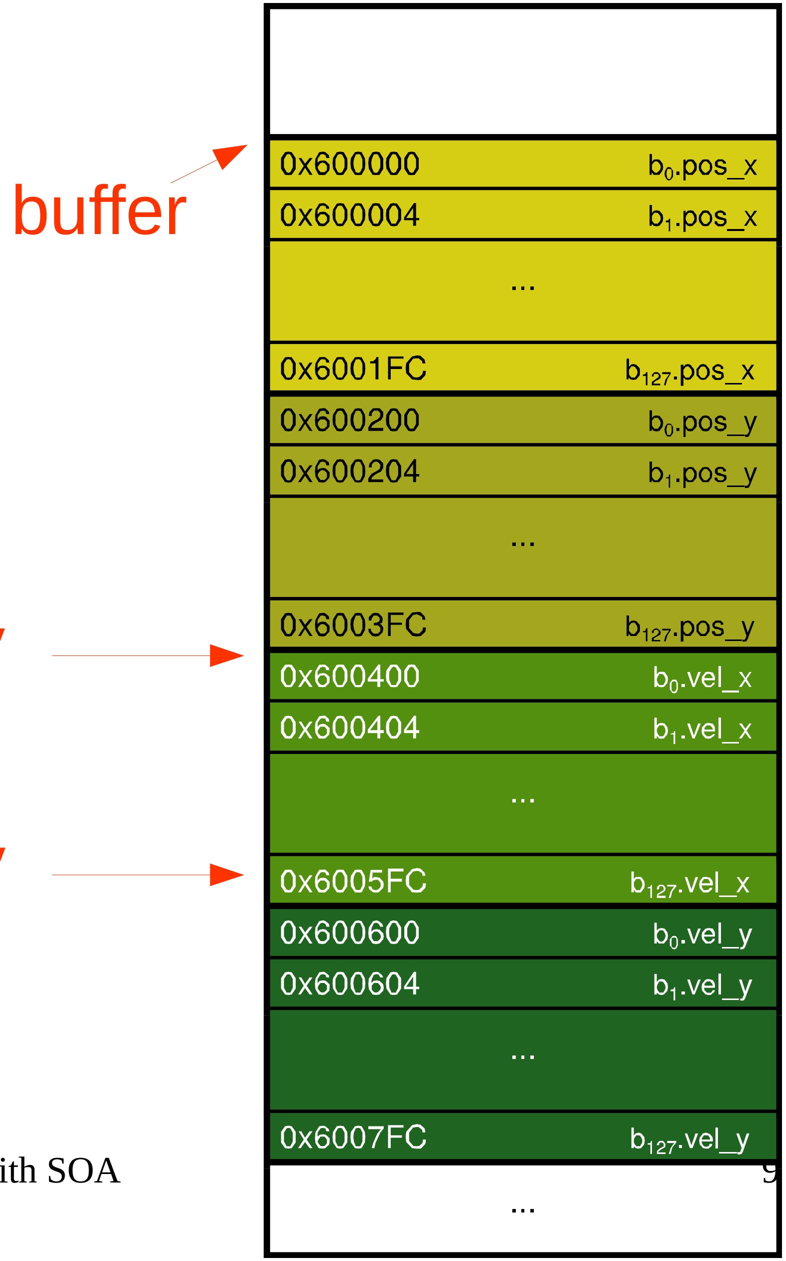


buffer

0x600000	b_0.pos_x
0x600004	b_1.pos_x
…	
0x6001FC	b_{127}.pos_x
0x600200	b_0.pos_y
0x600204	b_1.pos_y
…	
0x6003FC	b_{127}.pos_y
0x600400	b_0.vel_x
0x600404	b_1.vel_x
…	
0x6005FC	b_{127}.vel_x
0x600600	b_0.vel_y
0x600604	b_1.vel_y
…	
0x6007FC	b_{127}.vel_y
…	

Implementation Outline

e.g.: `float x = b127->vel_x;`



buffer

beginning of array

offset into array

0x600000	b_0.pos_x
0x600004	b_1.pos_x
...	
0x6001FC	b_{127}.pos_x
0x600200	b_0.pos_y
0x600204	b_1.pos_y
...	
0x6003FC	b_{127}.pos_y
0x600400	b_0.vel_x
0x600404	b_1.vel_x
...	
0x6005FC	b_{127}.vel_x
0x600600	b_0.vel_y
0x600604	b_1.vel_y
...	
0x6007FC	b_{127}.vel_y
...	

Implementation Outline

e.g.: `float x = b127->vel_x;`

float_ is a macro.

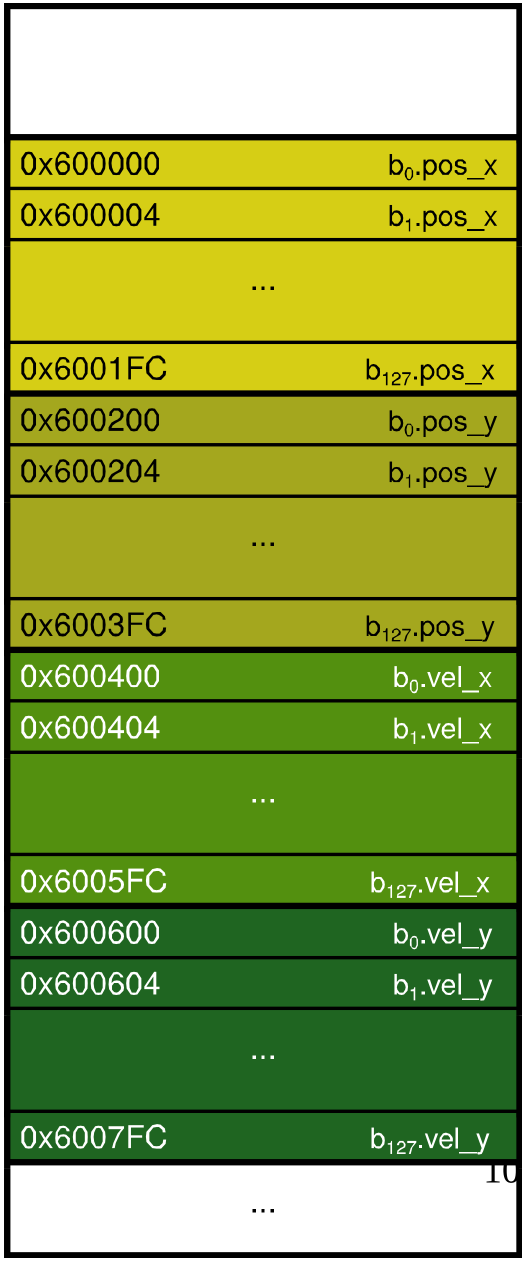
```
float_ vel_x;
=> Field<float, 8> vel_x;
```

Macro keeps track of field offsets.

buffer

beginning of array

offset into array



0x600000	b_{0}.pos_x
0x600004	b_{1}.pos_x
...	
0x6001FC	b_{127}.pos_x
0x600200	b_{0}.pos_y
0x600204	b_{1}.pos_y
...	
0x6003FC	b_{127}.pos_y
0x600400	b_{0}.vel_x
0x600404	b_{1}.vel_x
...	
0x6005FC	b_{127}.vel_x
0x600600	b_{0}.vel_y
0x600604	b_{1}.vel_y
...	
0x6007FC	b_{127}.vel_y
...	

Implementation Outline

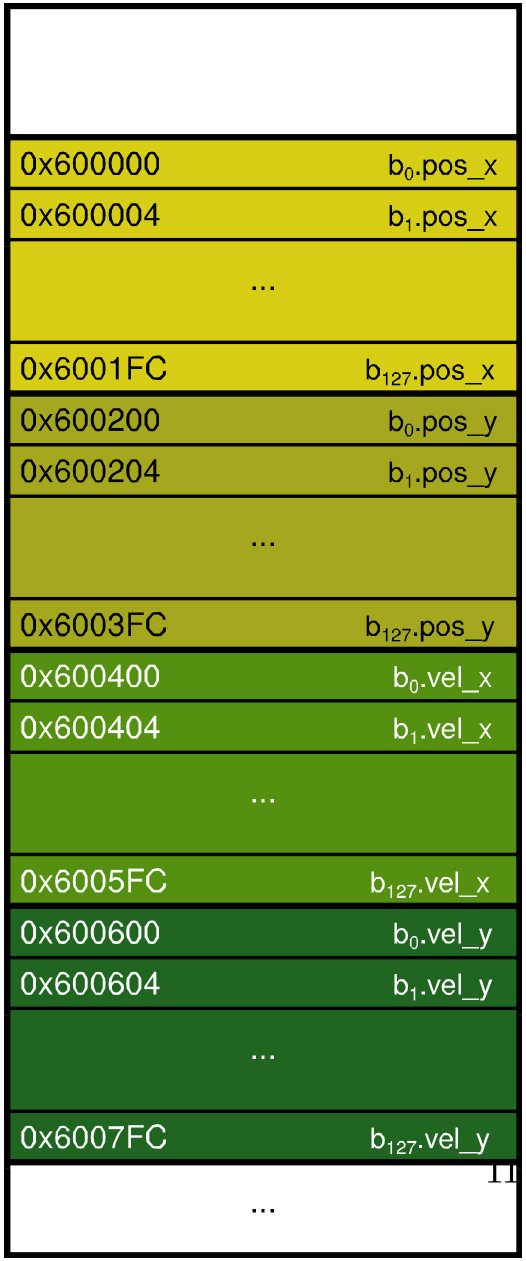
e.g.: `float x = b127->vel_x;`

float_ is a macro.

```
float_ vel_x;
=> Field<float, 8> vel_x;
```

“Fake” pointers encode IDs.

```
int Body::id() {
  return (int) this;
}
```



buffer

beginning of array

offset into array

0x600000	b_0.pos_x
0x600004	b_1.pos_x
…	
0x6001FC	b_{127}.pos_x
0x600200	b_0.pos_y
0x600204	b_1.pos_y
…	
0x6003FC	b_{127}.pos_y
0x600400	b_0.vel_x
0x600404	b_1.vel_x
…	
0x6005FC	b_{127}.vel_x
0x600600	b_0.vel_y
0x600604	b_1.vel_y
…	
0x6007FC	b_{127}.vel_y
…	

Performance Evaluation

```
float codegen_test(Body* ptr) {
    return ptr->vel_x;
}
```

Same performance (and assembly code) as in hand-written SOA code (gcc 5.4.0, clang 3.8)

→ Compilers can *understand* and optimize this code.
 (mainly constant folding)

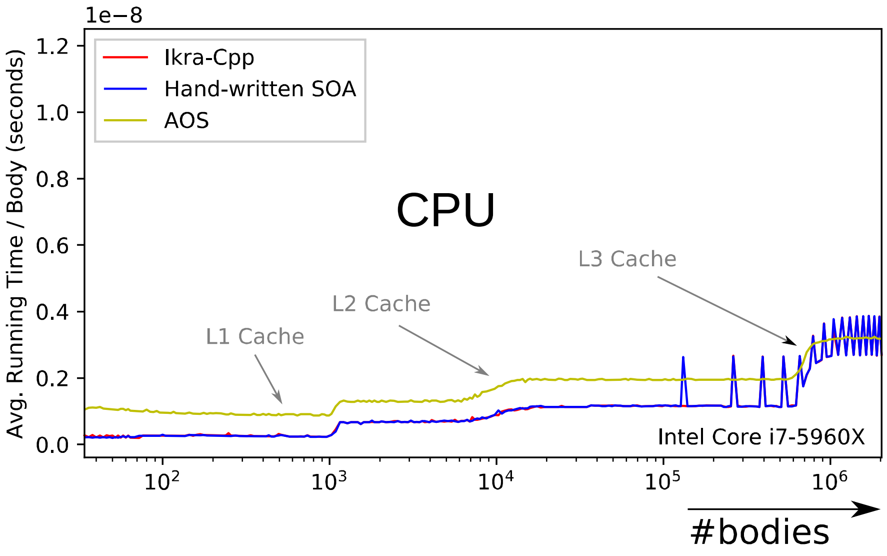
```
0000000000400690 <_Z11codegen_testP9Body>:
  400690:    8b 04 bd 60 10 60 00    mov    0x601060(,%rdi,4),%eax
  400697:    c3                      retq
  400698:    0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
  40069f:    00
```

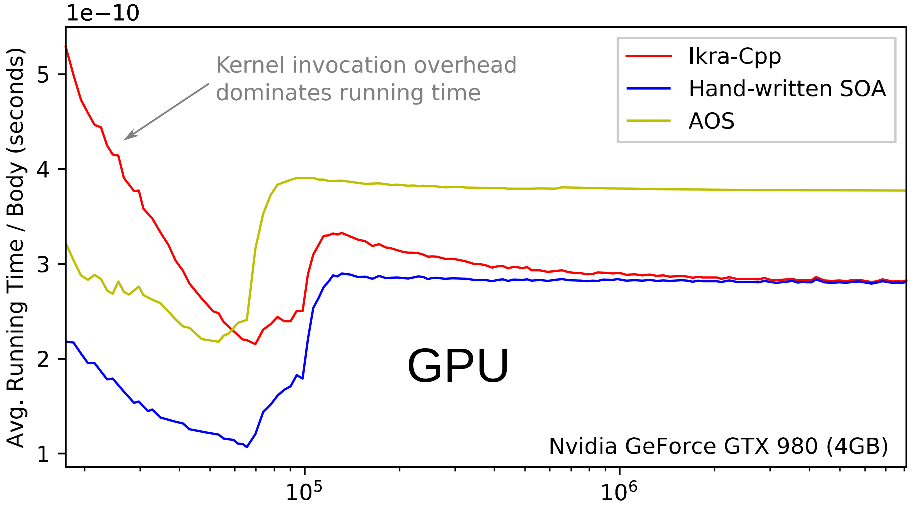
Performance Evaluation

```
forall(&Body::move, 0.5);
```

Compiler hints are necessary for auto-vectorization

- gcc: `constexpr` “hints”
- clang: No luck so far (problems with alias analysis)





Related Work

東京工業大学
Tokyo Institute of Technology

- ASX: Array of Structures eXtended

  Robert Strzodka. Abstraction for AoS and SoA Layout. In C++ GPU Computing Gems Jade Edition, pp. 429-441, 2012.

- SoAx

  Holger Homann, Francois Laenen. SoAx: A generic C++ Structure of Arrays for handling particles in HPC code. Comp. Phys. Comm., Vol. 224, pp. 325-332, 2018.

- Intel SPMD Compiler (ispc)

  Matt Pharr, William R. Mark. ispc: A SPMD compiler for high-performance CPU programming. In Innovative Parallel Computing (InPar), 2012.

Summary

- Embedded C++/CUDA DSL for SOA Layout
- OOP Features (pointers instead of IDs, member function calls, constructors, ...)
- Notation close to standard C++
- Implemented in C++, no external tools required
- Challenges/Future Work: Compiler optimizations (ROSE Compiler), inheritance, virtual function calls