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

Matthias Springer (Tokyo Institute of Technology)  https://github.com/prg-titech/ikra-cpp

Context: HPC with uniformly structured data (e.g., n-body simulation, traffic flow simulation)
Goal: SOA memory layout (good for caching, vectorization, parallelization) with C++ Notation
Pointers instead of IDs, Method Calls, new Keyword, Templates, Member of Object/Pointer Operator. Future work: virtual functions

Object Creation: Body *p = new Body(1.0, 2.0);
Body *q = Body::make(10, 1.0, 2.0);

Field Access: p->vel_x = p->vel_y = 1.5;
p->move(0.5);
forall(&Body::move, q, 10, 0.5);

class Body : public SOA<Body> {
public: INITIALIZE CLASS
float_ pos_x; float_ pos_y;
float_ vel_x; float_ vel_y;

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);

char buffer[128 * 16];
Large enough to store 128 objects (four float[128] arrays)

"Fake" Pointers encode Object IDs
There are various encoding techniques. Need to specify both an encoder (object construction) and a decoder (field access).

a) Zero Addressing: &obj_i = id

void* Body::operator new() {
    return (void*) size++;
} encoder

int field<T, idx, offset>::id() {
    Body* ptr = ((char*) this); decoder
    .idx* sizeof(field<T>);
    return (int) ptr;
} new keyword virtual functions

b) Valid Addressing: &obj_i buffer + id

c) First Field Addressing: &obj_i = buffer + sizeof(T) * id

Results & Main Insights
★ Field access (decoding object IDs + calculating memory addresses) is as efficient as array access in handwritten SOA code (stated mem. access).

Related Work


Make field<float> behave like a float
Implement implicit conversion and assignment operator.

T* field<T, idx, offset>::operator T&() {
    return *data_ptr();
}

Calculate memory location of field value
T* arr=(T*) (buffer+128*offset);
return arr + id();
(paddings needed for valid + first field addressing)