INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

intel experience what's inside™

INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

IMPROVE VECTORIZATION EFFICIENCY USING INTEL SIMD DATA LAYOUT TEMPLATE (INTEL SDLT)

Alex M Wells

Anoop Madhusoodhanan Prabha

Intel Corporation

November 2016

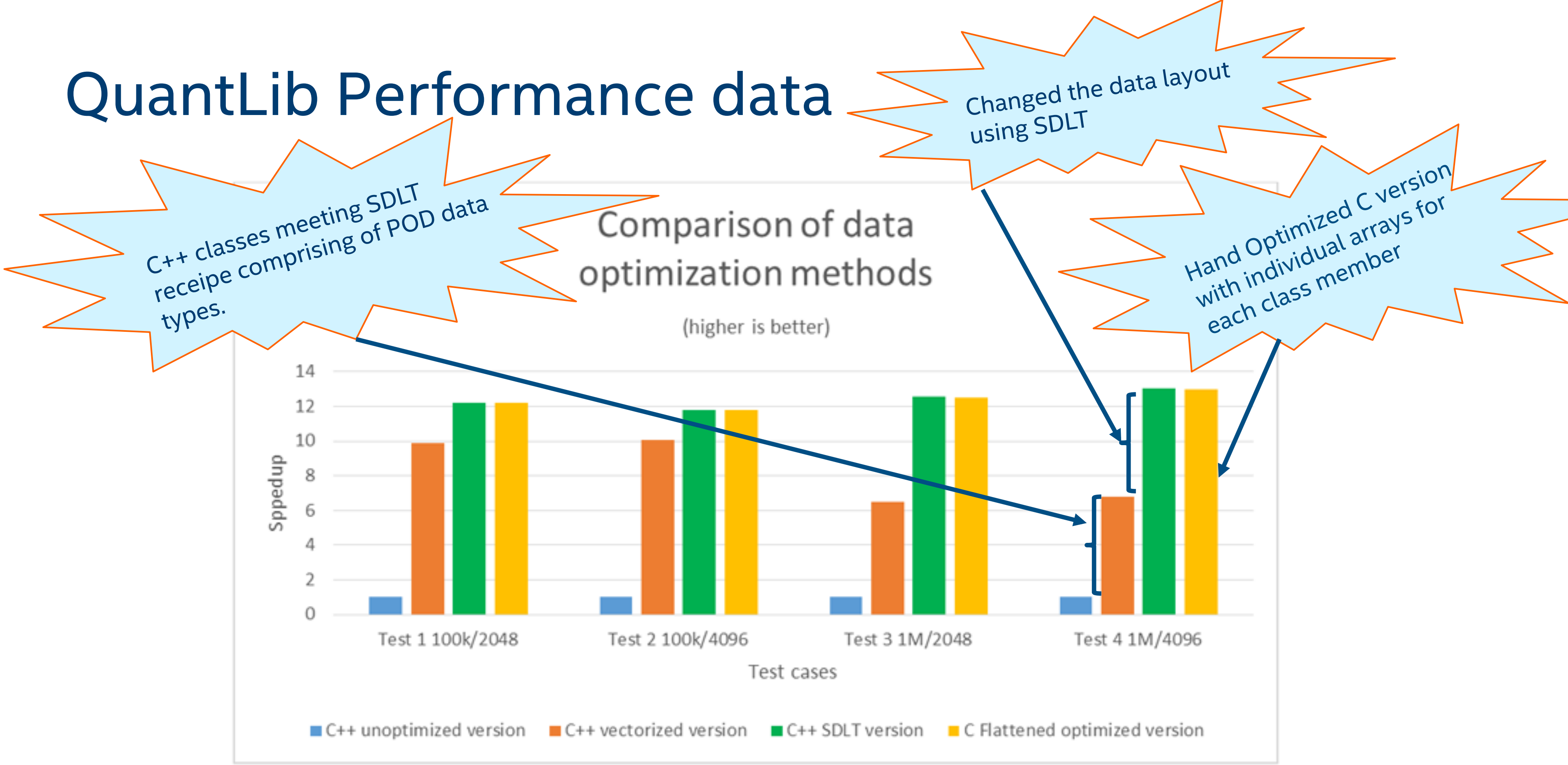
Long Story Short..

- Object-Oriented code involves modeling collections as Array of Structures (AoS).
- AoS data layout leads to lower vectorization efficiency.
- Intel SIMD Data Layout Template (Intel SDLT) helps developers stick to their Object-Oriented Design but still get better vectorization efficiency.

Agenda

- Performance Problems posed by Object-Oriented Design
- Why Intel SDLT?
- Components of Intel SDLT
- How to use Intel SDLT in your application?
- Q&A

QuantLib Performance data



https://software.intel.com/en-us/articles/data-layout-optimization-using-simd-data-layout-templates

Performance Problems posed by Object-Oriented Design

- Applications are designed as interaction between objects (Object-Oriented Design).
- Every real world entity is modeled as an object (user defined data type like struct or class).
- Collection of this above entity will become an Array of Structures (AoS).

```
struct YourStruct
{
    float x;
    float y;
    float z;
};
```

Heap Array

```
YourStruct * input = new YourStruct[count];
YourStruct * result = new YourStruct[count];
```

Stack Array

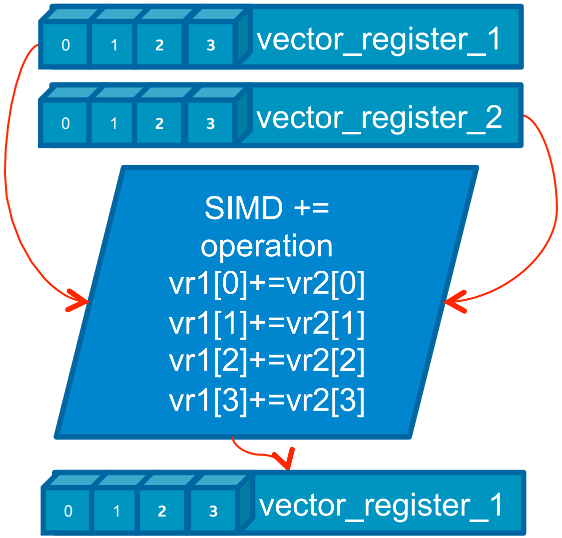
```
YourStruct input[count];
YourStruct result[count];
```

Vector

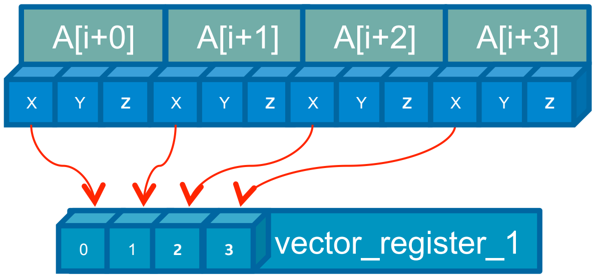
```
typedef std::vector<YourStruct> Container;
Container input(count);
Container result(count);
```

What's wrong with AoS?.... SIMD

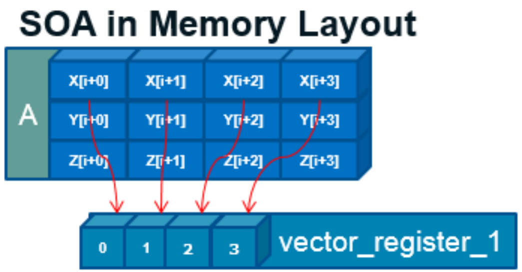
- SIMD – "Single Instruction Multiple Data"
- Vectorization with AOS in memory data layout requires multiple load/shuffle/insert or gather instructions.
- Increase in vector width demands more instructions for vector construction.
- Reduced CPU frequency in SIMD mode might not overcome SIMD improvement over the scalar code operating at a higher frequency.



AOS in Memory Layout



SIMD is effective with Unit Stride Access



- If memory layout has multiple instances of a data member adjacent in memory and aligned on a byte boundary matching the vector register width
  - Single load/store instruction to move the data into or out of a vector register
  - Many SIMD operations can reference an aligned unit-stride memory access as part of the instruction, avoiding a separate load/store instruction altogether
- A properly aligned Structure of Arrays (SOA) in memory data layout provides SIMD compatible Unit-Stride memory accesses
- SIMD efficiency & speedup can be restored

Issues with SoA integration

- Demands for change of the data structure and deviate from Object Oriented Design.
- Demands for change of C++ algorithms.
- Explicitly handle allocation/freeing of SOA arrays and make sure they they are aligned.

What is Intel SDLT?

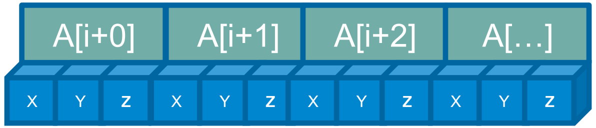
- A C++11 template library providing concepts of Containers, accessors and Offsets
  - Containers encapsulate the in memory data layout of an Array of “Plain Old Data“ objects.
  - SIMD loops use accessors with an array subscript operator (just like C++ arrays) to read from or write to the objects in the Containers.
  - Offsets can be embedded in accessors or applied to a Index passed to the accessors array subscript operator.
- Since these concepts are abstracted out, multiple concrete versions can exist and can encapsulate best known methods, thus avoiding common pitfalls in generating efficient SIMD code.

Why Intel SDLT?

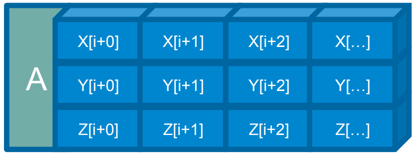
- SDLT provides a means to preserve the Array of Structure (AoS) interface for the developers but lays out the data in Structure of Array (SoA) format which is more SIMD friendly and increases the vectorization efficiency.
- SDLT provides 1D containers which provides the same interface as std::vector making the ease of integration and interoperability easy.
  - push_back, resize, etc.
  - iterator support: begin(), end(), etc.
- Works well with all STL algorithms
  - for_each, find, search, etc.
- Multi-Dimensional Container Support
  - Enabled in SDLT version 2 shipped with Intel C++ Compiler Version 17.0

SDLT Containers

- What if that std::vector could store data SOA internally while exposing an AOS view to the programmer?
  - Primary goal of SDLT Containers is to meet the requirement above.
- SDLT Containers abstract the in memory data layout to:
  - AOS (Array of Structures)
  - SOA (Structure of Arrays)



AOS (Array of Structures)



SOA (Structure of Arrays)

SDLT 1D container

```
typedef sdlt::soa1d_container<YourStruct> Container;
Container inputContainer(count);
Container resultContainer(count);
```

- Intent is data be kept in an SOA or ASA Container the entire time instead of converting from AOS.
- SDLT's container will internally store the members of YourStruct in a one dimensional "Structure of Arrays" (SOA) layout.
  - Places aligned arrays inside a single allocated buffer vs. a separate allocation per array
- Just like std::vector the Containers own the array data and its scope controls the life of that data.

SDLT Primitives

- How do the Containers discover the data members of your struct?
- C++ lacks compile time reflection, so the user must provide SDLT with some information on the layout of YourStruct.
- This is easily done with the SDLT_PRIMITIVE helper macro that accepts a struct type followed by a list of its data members.
  - A struct must be declared as a primitive before it is used as template parameter to a Container.

```
struct YourStruct
{
    float x;
    float y;
    float z;
};

struct AABB
{
    YourStruct topLeft;
    YourStruct bottomRight;
};

SDLT_PRIMITIVE(YourStruct, x, y, z)
SDLT_PRIMITIVE(AABB, topLeft, bottomRight)
```

SDLT Accessor

- To separate data ownership semantics from data access, a separate class called an accessor is used to access the transformed data that is owned by the Container.

```
Container::const_accessor<> input = inputContainer.const_access();
Container::accessor<> result = resultContainer.access();
```

- Use the C++11 keyword "auto" to let the compiler deduce the type.

```
auto input = inputContainer.const_access();
auto result = outputContainer.access();
```

SDLT Accessor Contd..

Embedded Offset

```
auto input = inputContainer.const_access();
auto input2 = inputContainer.const_access(256);
auto input3 = inputContainer.const_access(sdlt::aligned<8>(256));
auto input4 = inputContainer.const_access(sdlt::fixed<256>());
```

Subscript operator

```
void setAllValuesTo(
    Container::accessor iValues,
    const YourStruct &iDefautValue)
{
    for(int i=0; i < iValues.get_size_d1();
++i)
    {
        iValues[i] = iDefaultValue;
    }
}
```

SDLT Accessor Contd..

- The subscript operator[index] returns a Proxy Object
- The main use of the Proxy Objects is to import/export data to/from a local variable
- Can assign local stack instance of YourStruct to the Proxy

```
YourStruct result = …
iValues[index] = result;
```

- Can retrieve YourStruct from the Proxy to a local stack object

```
iValues[index].y() = new_y_value;
YourStruct local = iValues[index];
```

- SDLT's design makes use of local objects and the compiler's dead code elimination features.
- Overloaded +=, -=, *=, etc. operators.

Multiple Dimensions with sdlt::n_container

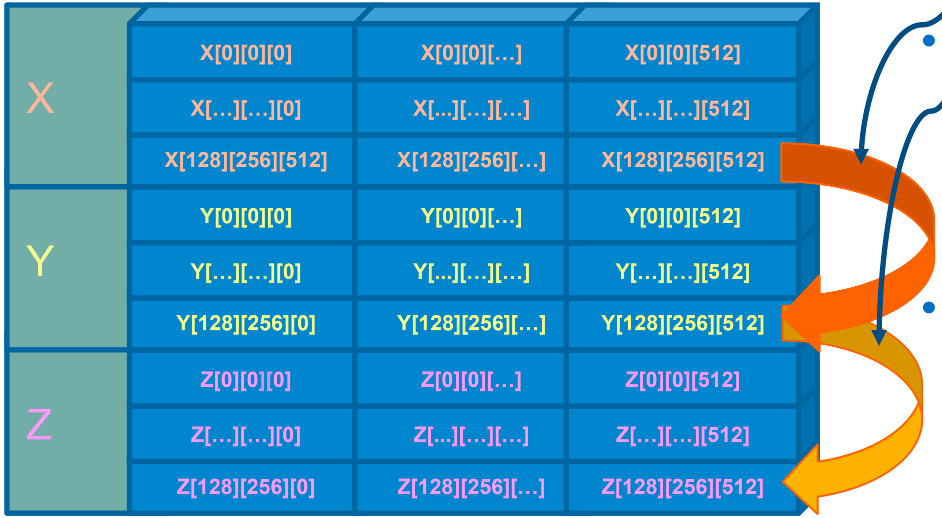
```
using namespace sdlt;

auto shape = n_extents[128][256][512];
typedef n_container<YourStruct, layout::soa, decltype(shape)> Container3d;

Container3d input(shape), output(shape);
auto inputs = input.const_access();
auto outputs = output.access();

for(int z = 0; z < 128; ++z) {
    for(int y = 0; y < 256; ++y) {
        #pragma omp simd
        for(int x = 0; x < 512; ++x) {
            YourStruct val = inputs[z][y][x];
            YourStruct result = … // compute result
            outputs[z][y][x] = result;
        }
    }
}
```

- The shape is described with the sdlt::n_extents generator object
- Use accessors with multiple array subscript operators, just like multi-dimensional C arrays

Issues with Large Arrays with SOA

Memory layout of a 3d SOA

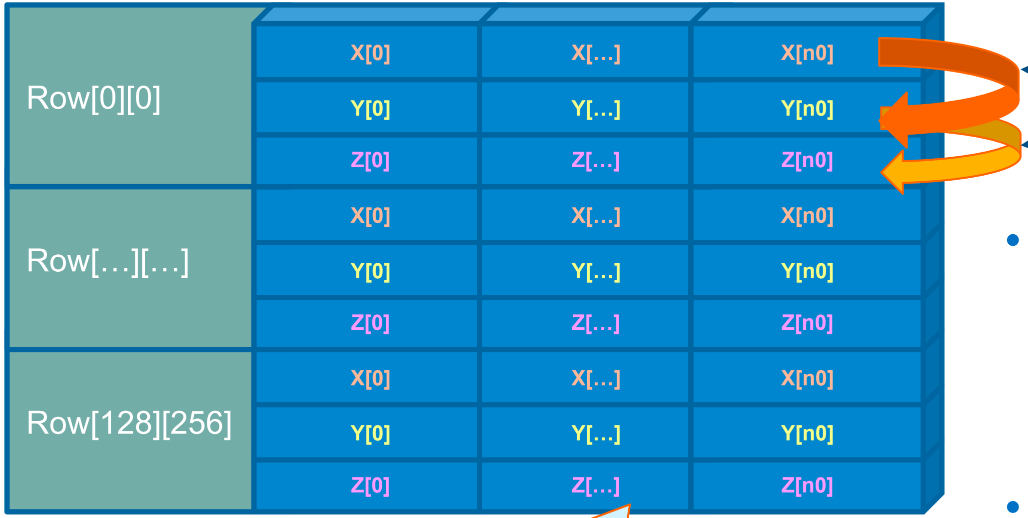
```
n_container<YourStruct,
            layout::soa,
            decltype(n_extents[128][256][512])>
```



- For any given element:
  - a[z_index][y_index][x_index]
- Distance between data members:
  - >= 65mb ~ 128*256*512*sizeof(data_member)
- Each data member:
  - Possibly in different virtual memory pages.
  - Appears as a separate data stream to hardware prefetchers
- As # data members or hyper threads increase, so does
  - DTLB pressure
  - Hardware prefetcher pressure

SOA Per Row Memory Layout

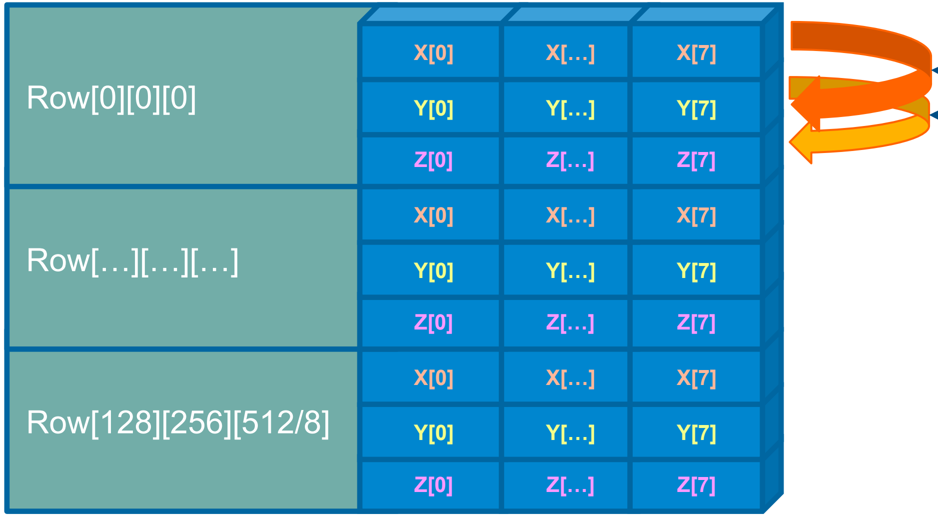
```
n_container<YourStruct,
            layout::soa_per_row,
            decltype(n_extents[128][256][512])>
```



- For any given element:
  - a[z_index][y_index][x_index]
- Distance between data members:
  - >= 2kb ~ 512*sizeof(data_member)
- Each data member:
  - Likely in same virtual memory page.
  - Likely appears as a separate data stream to hardware prefetchers
- As # data members or hyper threads increase
  - Hardware prefetcher pressure

Adding Xtra Blocking Dimension

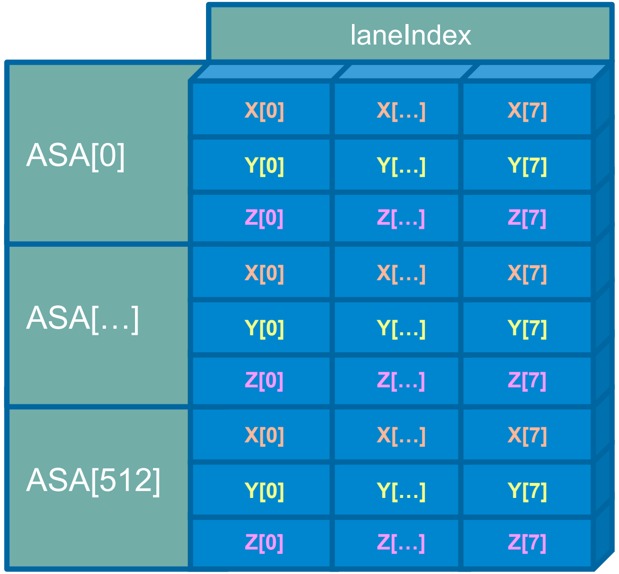
```
constexpr int vec_width=8;
n_container<YourStruct,
            layout::soa_per_row,
            decltype(n_extents[128][256][512/vec_width][vec_width])>
```



- For any given element:
  - a[z_index][y_index][x_index/8][x_index%8]
- Distance between data members:
  - >= 32b ~ 8*sizeof(data_member)
- Each data member:
  - **IS in the same same virtual memory page.**
  - Appears as a segment of a single linear data stream to hardware prefetchers

Hardware prefetcher friendly

By Hand, Combine AoS with Fixed Size SoA




```
constexpr int lane_count=8;
struct SimdYourStruct {
    float x[lane_count];
    float y[lane_count];
    float z[lane_count];
} __attribute__((aligned(32)));

int count = 4096;
int structCount = count/lane_count;
SimdYourStruct inputASA[structCount];
SimdYourStruct outputASA[structCount];

for(int structIndex=0; structIndex < structCount; ++structIndex) {
    #pragma omp simd
    for(int laneIndex=0; laneIndex < lane_count; ++laneIndex) {
        YourStruct val;
        val.x = inputASA[structIndex].x[laneIndex];
        val.y = inputASA[structIndex].y[laneIndex];
        val.z = inputASA[structIndex].x[laneIndex];
        YourStruct result = … // compute result
        outputASA[structIndex].x[laneIndex] = val.x;
        outputASA[structIndex].y[laneIndex] = val.y;
        outputASA[structIndex].x[laneIndex] = val.z;
    }
}
```

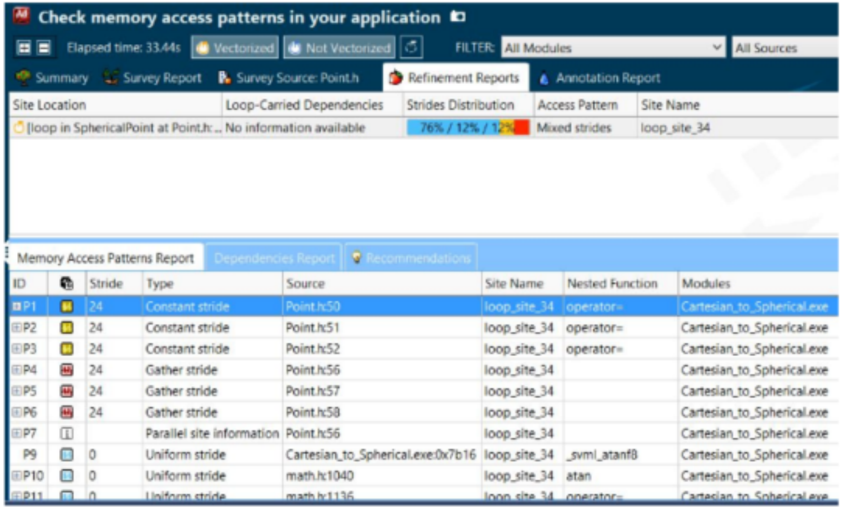
With SDLT, Combine AoS with Fixed Size SoA

```
constexpr int lane_count=8;
int count = 4096;
auto shape = n_extents[count/lane_count][fixed<lane_count>()];
typedef n_container<YourStruct, layout::soa_per_row, decltype(shape)> Container;

Container inputASA(shape), outputASA(shape);
auto inputs = input.const_access();
auto outputs = output.access();

for(int structIndex=0; structIndex < extent_d<0>(inputs); ++structIndex) {
    #pragma omp simd
    for(int laneIndex=0; laneIndex < extent_d<1>(inputs); ++laneIndex) {
        YourStruct val = inputs[structIndex][laneIndex];
        YourStruct result = … // compute result
        outputs[structIndex][laneIndex] = result;
    }
}
```

- sdlt::fixed<int> represents an integral constant known at compile time
- Template function sdlt::extent_d<int> determines the extent of a dimension for a multi-dimensional object

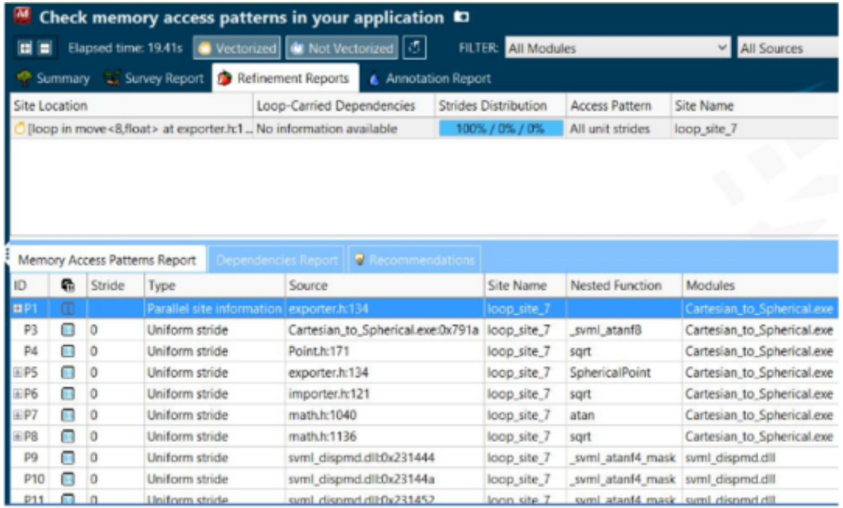
Before Intel SDLT enabling

```
class CartesianPoint{
public:
    float x, y, z;
    CartesianPoint(){
        x = y = z = 0.0f;
    }
    explicit CartesianPoint(float x1, float y1, float z1){
        x = x1;
        y = y1;
        z = z1;
    }
    CartesianPoint(const CartesianPoint& other)
    : x(other.x)
    , y(other.y)
    , z(other.z)
    {
    }
    CartesianPoint& operator=(float n){
        x = y = z = n;
        return *this;
    }
};
class CartesianPointImage{
public:
    vector<CartesianPoint> cartpoints;
    int num_of_points() const { return static_cast<int>(cartpoints.size()); }
    CartesianPointImage(size_t num_of_elements){
        cartpoints.resize(num_of_elements);
        for (unsigned int i = 0; i < num_of_elements; i++)
        {
            cartpoints[i] = CartesianPoint(i, i, i);
        }
    }
};
```



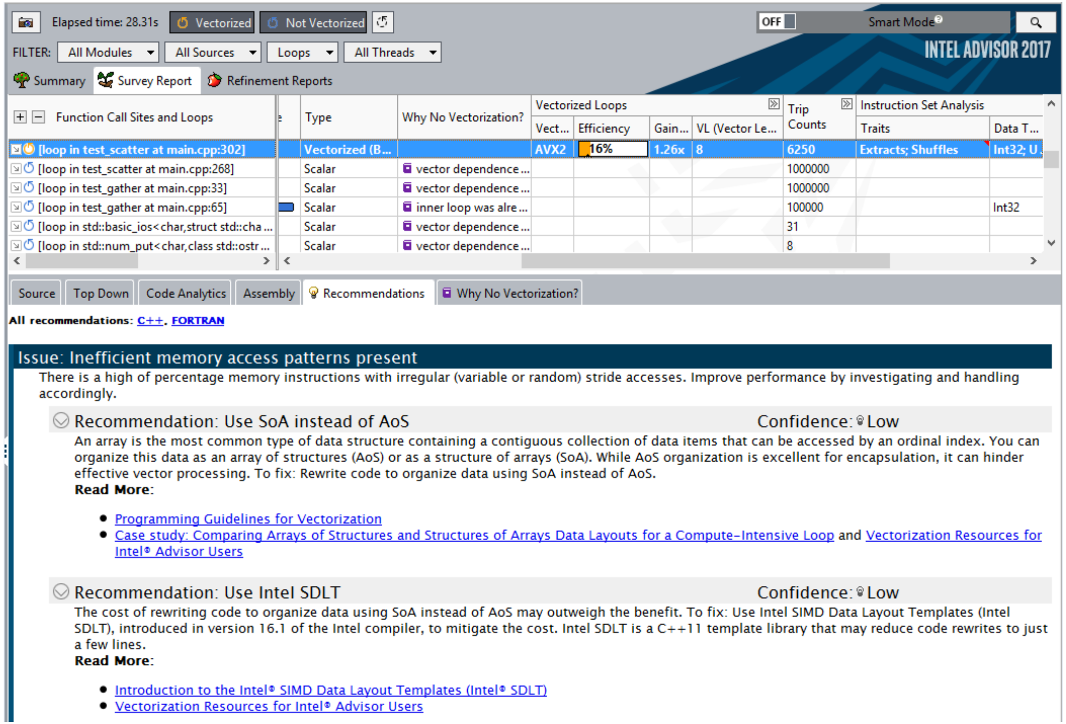
After Intel SDLT enabling

```
class CartesianPointImage{
public:
    CartesianPointContainer cartpoints; //This is the key change
    int num_of_points() const { return static_cast<int>(cartpoints.size()); }
    CartesianPointImage(size_t num_of_elements){
        cartpoints.resize(num_of_elements);
        auto a = cartpoints.begin();
        for (unsigned int i = 0; i < num_of_elements; i++)
        {
            a[i] = CartesianPoint(i, i, i);
        }
    }
};
```



http://bit.ly/intelsdlt-wp2

Vector Advisor's Recommendation for Intel SDLT



Case Study

- Efficient SIMD in Animation with SDLT and Data preconditioning
- DreamWorks Animation (DWA): How We Achieved a 4x Speedup of Skin Deformation with SIMD

Resources

Intel SIMD Data Layout Template Info

- Introducing the Intel SIMD Data Layout Template (Intel SDLT) to boost efficiency in your vectorized C++ code
  - http://bit.ly/intelsdlt
- Introduction to the Intel SDLT
  - https://software.intel.com/en-us/node/684050
- Averaging Filter with Intel SDLT
  - http://bit.ly/intelsdlt-wp1
- Intel Xeon Phi Processor High Performance Programming book, Chapter 11 – Vectorization with SDLT, ISBN: 0128091959
- Boosting the performance of Cartesian to Spherical co-ordinates conversion using Intel SDLT
  - http://bit.ly/intelsdlt-wp2

Code Modernization Links

- Modern Code Developer Community
  - https://software.intel.com/modern-code
- Intel Code Modernization Enablement Program
  - https://software.intel.com/code-modernization-enablement
- Intel Parallel Computing Centers
  - https://software.intel.com/ipcc
- Technical Webinar Series Registration
  - http://bit.ly/spring16-tech-webinars
- Intel Parallel Universe Magazine
  - https://software.intel.com/intel-parallel-universe-magazine

Questions?

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

THANK YOU FOR YOUR TIME

Alex M Wells & Anoop Madhusoodhanan Prabha

www.intel.com/hpcdevcon

experience
what's inside™

Possible Overhead When Storing Objects

- We are assigning a newBoundary object to a container.
- We only want to change the "y" component of the bounds,
  - Because we can only import entire objects
    - We must initialize a new Point3ds
    - Transfer the entire object into the container.
      - Will include the "x" and "z" components despite the fact they haven't changed.
    - Because it’s and assignment, the compiler can't figure out the values haven't changed.
      - Perhaps another thread had changed the values, and we are reassigning them back.
- The point is that it won't eliminate the assignments to the "x" and "z" inside the container.

```
for(int i=0; i < count; ++i) {
    const Point3ds point = points[i];
    const Point3ds boundary = bounds[i];
    if( point.y > boundary.y) {
        Point3ds newBoundary(boundary.x, newpoint.y, boundary.z);
        bounds[i] = newBoundary;
    }
}
```

SDLT Proxy Objects Provide Interface to Data Members

- The proxy objects SDLT returns from the [i] operator provide an interface to access the individual data members of the primitive.
- The interface provides a method using each data member's name and returns a proxy to that data member for element [i] inside the Container.

```
for(int i=0; i < count; ++i) {
    const Point3ds point = points[i];
    const Point3ds boundary = bounds[i];
    if( point.y > boundary.y) {
        bounds[i].y() = point.y;
    }
}
```

- Now only the "y" component will be updated and the loop is much more efficient.