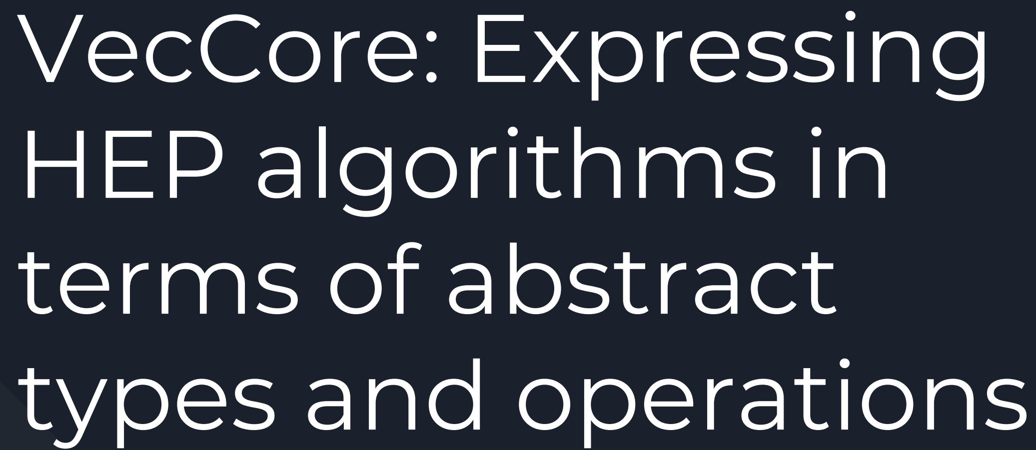
VecCore: Expressing HEP algorithms in terms of abstract types and operations

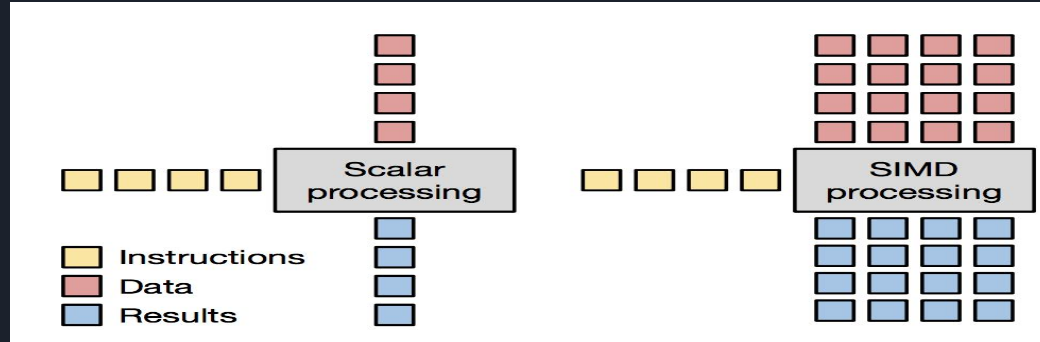


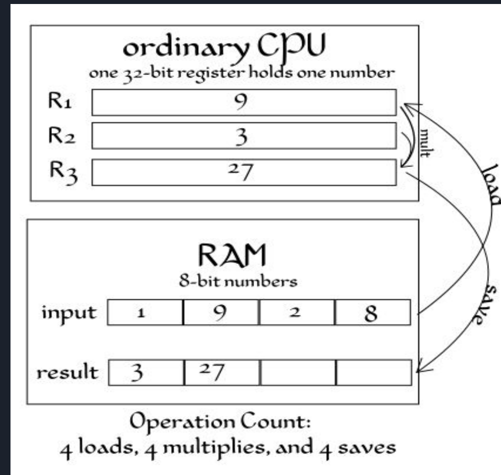
Joint WLCG & HSF workshop
Napoli 2018
andrei.gheata@cern.ch

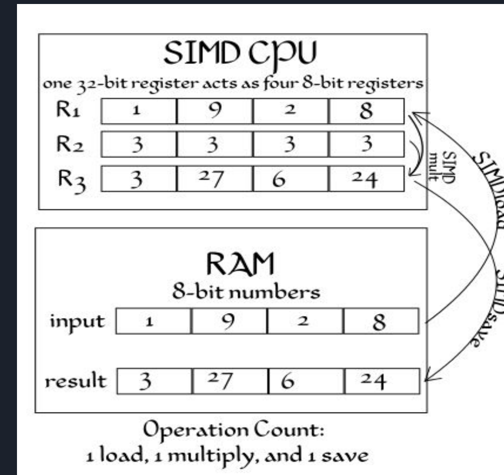
Menu

- Vectorization, why should we bother?
  - Hardware, evolution, promises, caveats
- Vectorization techniques, what to choose…
  - Auto-vectorization - can we go simple?
  - VecCore in this landscape
- Is HEP code suitable?
  - Data and algorithm reshaping
  - Examples: dealing with inner and outer loops
  - Vector-friendly HEP libraries?
- Can we vectorize the event loop? 😏 😋 😂
  - The top-down approach

SIMD: the principle





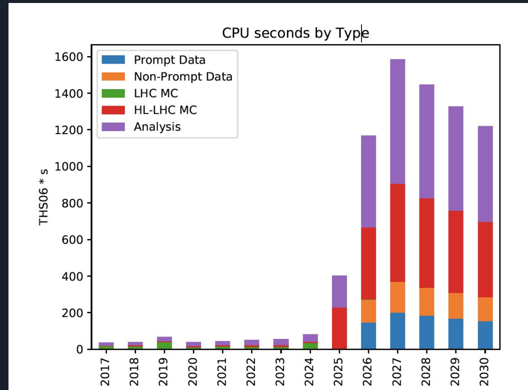


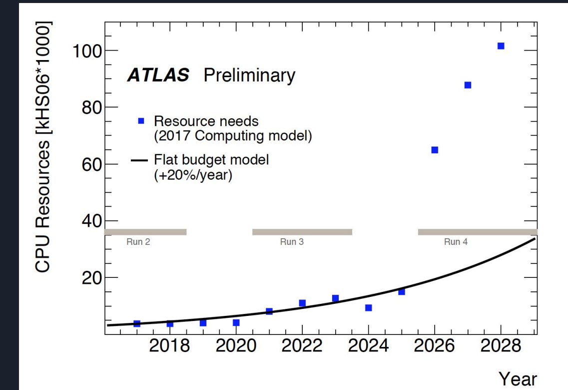
B[i] = 3 * A[i]

wikipedia.org

Do we need it?

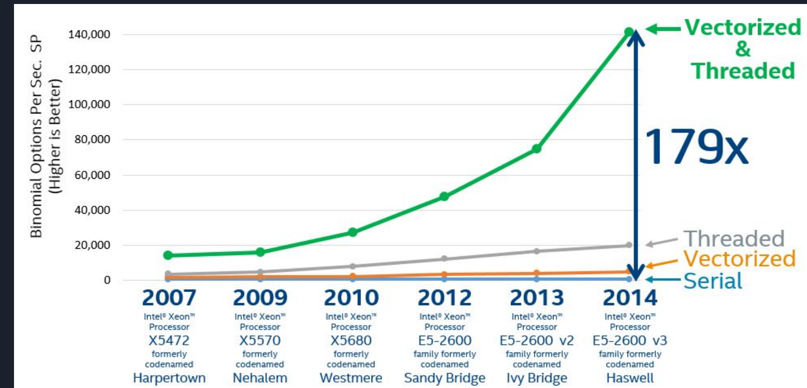
CMS and Atlas estimated CPU needs for HL-LHC (source: CWP)





- Estimated ~10x CPU needs for the HL-LHC era
- Some of this factor must come from better utilisation of the existing hardware

1. Reviewing the software can reveal important optimization potential
2. Vectorization is one of the few methods to increase throughput
    - 2 implies 1



Evolution of vector architectures

- Process vectors of data one word at a time, executing single instructions for pipelines of processors
  - ILLIAC IV (1970)'s: 64 ALUs giving ~100 MFLOPS
  - Supercomputers: STAR-100 (CDC), ASC (Texas Instruments) -> pipelines for vector operations (data intensive tasks, such as fluid dynamics)
  - Cray-1 : first architecture based on registers - multi pipeline arch.
- Modern machines "commodity SIMD" from the 90's. Vector instruction sets operating on registers
  - MMX, SSE -> AVX2, AVX-512 (Intel)
  - AltiVec (Power architectures)
  - NEON (ARM)
  - GPU - pipelines driven by compute kernels

wikiperdia.org

Cray-1 (1976)
8x64 bit vector registers
160 MFLOPS, ~8M USD w/o disk
5.5 tons, 115 kW

How do we get it?

- Auto-vectorization
  - Compiler optimization converting repetitive scalar instructions (loops) to SIMD code
- Compiler pragmas
  - Code annotations persuading the compiler into vectorizing
  - OpenMP, CilkPlus
  - May not preserve exact scalar behavior
- SIMD libraries
  - VCL, Vc, UME::SIMD, VecCore
  - Explicit programming using specific vector types and operations
- Compiler intrinsics
  - Built-in inline compiler functions accessing architecture-specific vector instructions
- Assembly
  - The really low-level stuff on top of HW implementation

Programability | Performance | Portability

```
float a[N], b[N], c[N];

for (int i = 0; i < N; i++)
  a[i] = b[i] * c[i];
```

```
float a[N], b[N], c[N];

#pragma omp simd
#pragma ivdep
for (int i = 0; i < N; i++)
  a[i] = b[i] * c[i];
```

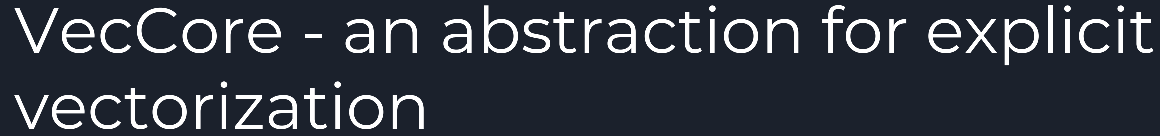
```
#include <VecCore/VecCore>
using Float_v =
   backend::VcVector::Float_v;
Float_v a = b * c;
```

```
#include <x86intrin.h>
__m256 a, b, c;

a = _mm256_mul_ps(b, c);
```

```
asm volatile("vmulps %ymm1,
%ymm0");
```

What may prevent vectorization?

- Compiler must prove that the operation leads to same result -> Not always easy
  - Loop data dependencies, pointer aliasing, non-inline function calls, early returns, nested control flow, conditional recursions, …
  - HEP code has lots of those...
- Compilers may add runtime checks (tails, loop size), vectorize parts of the loop, or just refuse to vectorize
  - Performance can vary wildly among compilers (https://godbolt.org)
  - And doing small changes in a large code base may silently disable auto-vectorization
- In many cases there is no loop, or at too high level (events, tracks)



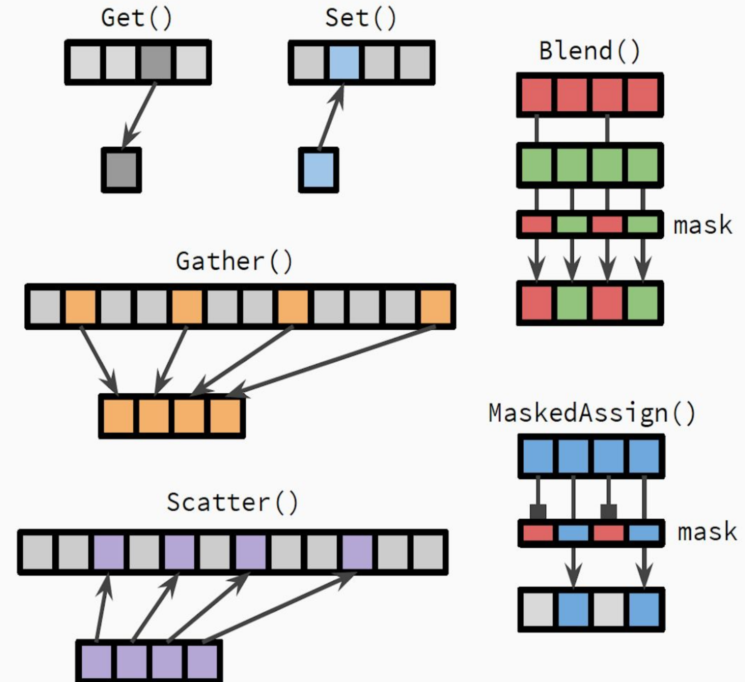
VecCore - an abstraction for explicit vectorization

- VecCore is an evolution of the idea of “backends” that existed in VecGeom
  - Coherent set of abstract vector types and operations implemented by the backend
  - J. DeFineLicht, P. Canal, S.Wenzel
- Extracted into today’s standalone VecCore
  - Unification and formalization of the backend API
  - New operations (e.g. Gather/Scatter)
  - G. Amadio, P. Canal, S. Wenzel
- Hosted by GitHub ROOT project and used as ROOT external
  - https://github.com/root-project/veccore
- Hardware architectures supported
  - Intel® Architectures (SSE4.2, AVX/AVX2, AVX-512)
  - Nvidia GPUs via CUDA
  - ARM neon and PowerPC Altivec *supported with the scalar backend!*

VecCore operations

G. Amadio, ACAT'17

```
namespace vecCore {

template <typename T> struct TypeTraits;
template <typename T> using Mask   = typename TypeTraits<T>::MaskType;
template <typename T> using Index  = typename TypeTraits<T>::IndexType;
template <typename T> using Scalar = typename TypeTraits<T>::ScalarType;

// Vector Size
template <typename T> constexpr size_t VectorSize();

// Get/Set
template <typename T> Scalar<T> Get(const T &v, size_t i);
template <typename T> void Set(T &v, size_t i, Scalar<T> const val);

// Load/Store
template <typename T> void Load(T &v, Scalar<T> const *ptr);
template <typename T> void Store(T const &v, Scalar<T> *ptr);

// Gather/Scatter
template <typename T, typename S = Scalar<T>>
T Gather(S const *ptr, Index<T> const &idx);

template <typename T, typename S = Scalar<T>>
void Scatter(T const &v, S *ptr, Index<T> const &idx);

// Masking/Blending
template <typename M> bool MaskFull(M const &mask);
template <typename M> bool MaskEmpty(M const &mask);

template <typename T> void MaskedAssign(T &dst, const Mask<T> &mask, const T &src);
template <typename T> T Blend(const Mask<T> &mask, const T &src1, const T &src2);

} // namespace vecCore
```



https://github.com/root-project/veccore/blob/master/doc/API.md

Using VecCore

Data extraction from higher-level loop

```
algorithm( const scalar_t &input);
```

```
algorithm( const SOA<scalar_t>  &input_v);
```

Scalar interface

Vector interface

Internal vectorization

```
template<typename Real_v>
void Implementation( const Scalar<Real_v> &input,
                     Scalar<Real_v> &output )
{
  // Algorithm vectorizing on internal loops, input always scalar
  for (size_t i=0; i<N; i += VectorSize<Real_v>()) {
    // algorithm filling data in VecCore vector types and
    // using VecCore  operations
  }
}
```

Real_v aliases a VecCore vector type, e.g. vecCore::backend::VcVector::Real_v

Multiple-input vectorization

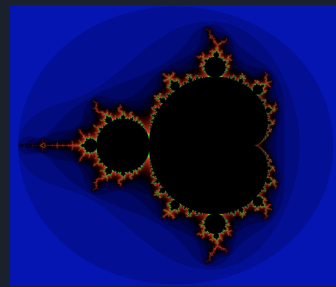
```
template<typename Real_v>
void Implementation( const Real_v &input,
                     Real_v & output )
{
  Algorithm operating of generic type, scalar or vector
  // algorithm using VecCore operations on the generic type (scalar
  // or vector)
}
```

Real_v aliases an arbitrary scalar/vector type

Transforming naturally vectorizable scalar algorithms

Writing generic algorithms for scalar or vector inputs (common “kernel”)

Internal vectorization example: the Mandelbrot set



Iterate $f(z) = z^2 + c$ N times and check if z diverges

scalar

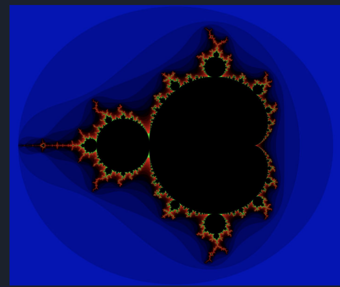
```
template<typename T>
void mandelbrot(T xmin, T xmax, size_t nx,
                T ymin, T ymax, size_t ny,
                size_t max_iter, unsigned char *image)
{
    T dx = (xmax - xmin) / T(nx);
    T dy = (ymax - ymin) / T(ny);
    for (size_t i = 0; i < nx; ++i) {
        for (size_t j = 0; j < ny; ++j) {
            size_t k = 0;
            T x = xmin + T(i) * dx, cr = x, zr = x;
            T y = ymin + T(j) * dy, ci = y, zi = y;
            do {
                x = zr*zr - zi*zi + cr;
                y = 2.0 * zr*zi + ci;
                zr = x;
                zi = y;
            } while (++k < max_iter && (zr*zr+zi*zi < 4.0));
            image[ny*i+j] = k;
        }
    }
}
```

ACAT'17

VecCore

```
template<typename T>
void mandelbrot_v(Scalar<T> xmin, Scalar<T> xmax, size_t nx,
                  Scalar<T> ymin, Scalar<T> ymax, size_t ny,
                  Scalar<Index<T>> max_iter, unsigned char *image)
{
    T iota;
    for (size_t i = 0; i < VectorSize<T>(); ++i) Set<T>(iota, i, i);
    T dx = T(xmax - xmin) / T(nx);
    T dy = T(ymax - ymin) / T(ny), dyv = iota * dy;
    for (size_t i = 0; i < nx; ++i) {
        for (size_t j = 0; j < ny; j += VectorSize<T>()) {
            Scalar<Index<T>> k{0};
            T x = xmin + T(i) * dx, cr = x, zr = x;
            T y = ymin + T(j) * dy + dyv, ci = y, zi = y;
            Index<T> kv{0};
            Mask<T> m{true};
            do {
                x = zr*zr - zi*zi + cr;
                y = T(2.0) * zr*zi + ci;
                MaskedAssign<T>(zr, m, x);
                MaskedAssign<T>(zi, m, y);
                MaskedAssign<Index<T>>(kv, m, ++k);
                m = zr*zr + zi*zi < T(4.0);
            } while (k < max_iter && !MaskEmpty(m));
        for (size_t k = 0; k < VectorSize<T>(); ++k)
            image[ny*i+j+k] = (unsigned char) Get(kv, k);
}}}
```

Internal vectorization example: the Mandelbrot set



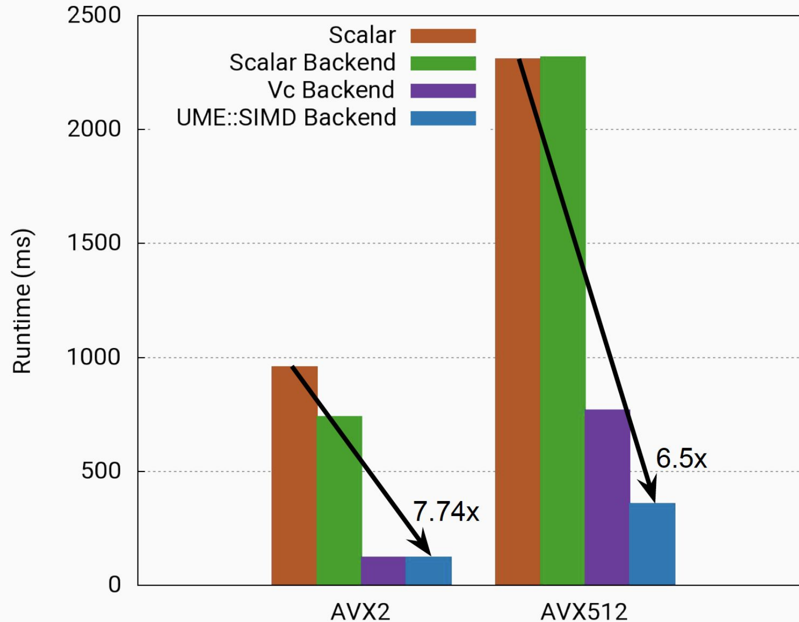
Iterate f(z) = z² + c N times and check if z diverges

scalar

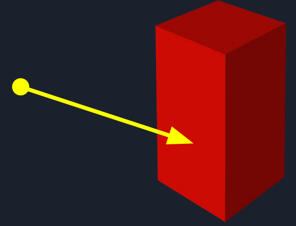
```
template<typename T>
void mandelbrot(T xmin, T xmax,
                T ymin, T ymax,
                size_t max_iter,
{
    T dx = (xmax - xmin) / T(nx);
    T dy = (ymax - ymin) / T(ny);
    for (size_t i = 0; i < nx; ++i) {
        for (size_t j = 0; j < ny; ++j) {
            size_t k = 0;
            T x = xmin + T(i) * dx, cr =
            T y = ymin + T(j) * dy, ci =
            do {
                x = zr*zr - zi*zi + cr;
                y = 2.0 * zr*zi + ci;
                zr = x;
                zi = y;
            } while (++k < max_iter &
            image[ny*i+j] = k;
        }
    }
}
```

VecCore

```
T> xmax, size_t nx,
T> ymax, size_t ny,
iter, unsigned char *image)

Set<T>(iota, i, i);

* dy;

<T>()) {

i = y;

, ++k);

pty(m));
+k)
et(kv, k);
```



Multiple input vectorization example: VecGeom box



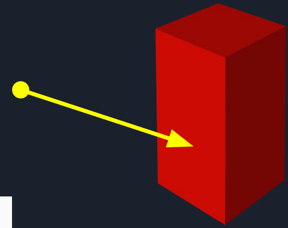
scalar

```
template <typename T>
static void DistanceToIn(UnplacedStruct_t const &box,
                         Vector3D<T> const &point,
                         Vector3D<T> const &dir, T &distance)
{
  using Vector_t = Vector3D<T>;
  const Vector_t invdir(1.0 / NonZero(direction[0]),
                        1.0 / NonZero(direction[1]),
                        1.0 / NonZero(direction[2]));
  const Vector_t signDir(Sign(dir[0]), Sign(dir[1]), Sign(dir[2]));
  const Vector_t tempIn  = -signDir * box.fDimensions - point;
  const Vector_t tempOut = signDir * box.fDimensions - point;
  // add a check for point on exit surface
  const T absOrthogOut = Abs((signDir * tempOut).Min());
  const T distOut = (tempOut * invDir).Min();
  // distIn calculation
  distance = (tempIn * invDir).Max();
  if (distance >= distOut ||
      distOut <= kTolerance) ||  absOrthogOut <= kHalfTolerance)
     distance = InfinityLength<T>();
}
```

VecCore

```
template <typename Real_v>
static void DistanceToIn(UnplacedStruct_t const &box,
                         Vector3D<Real_v> const &point,
                         Vector3D<Real_v> const &dir,Real_v &distance)
{
  using Vector_t = Vector3D<Real_v>;
  const Vector_t(Real_v(1.0) / NonZero(direction[0]),
                 Real_v(1.0) / NonZero(direction[1]),
                 Real_v(1.0) / NonZero(direction[2]));
  const Vector_t signDir(Sign(dir[0]), Sign(dir[1]), Sign(dir[2]));
  const Vector_t tempIn  = -signDir * box.fDimensions - point;
  const Vector_t tempOut = signDir * box.fDimensions - point;
  // add a check for point on exit surface
  const Real_v absOrthogOut = Abs((signDir * tempOut).Min());
  const Real_v distOut = (tempOut * invDir).Min();
  // distIn calculation
  distance = (tempIn * invDir).Max();
  vecCore::Mask<Real_v> bad = distance >= distOut ||
     distOut <= kTolerance) ||  absOrthogOut <= kHalfTolerance;
  MaskedAssign(distance, bad,   InfinityLength<Real_v>());
}
```

Multiple input vectorization example: VecGeom box



Box Benchmark – Intel® Core™ i7-6700 CPU 3.40GHz (Skylake)

Runtime (ms)

5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0

Inside
Contains
SafetyToIn
SafetyToOut
DistanceToIn
DistanceToOut

USolids (No SIMD)
USolids
Scalar API (No SIMD)
Scalar API (Scalar)
Vector API (No SIMD)
Vector API (Scalar)
Vector API (Vc)
Vector API (UMESIMD)
Geant4

Code becomes: portable across CPU/GPGPU, SIMD-portable, reusable for scalar/vector scenarios…

… not mentioning that transforming the code to vectorize makes generally better scalar code...

Lower is better

Proposal approved at the last C++ committee meeting

To be published in ~2 weeks time scale

Likely to be available in C++20

Adopting VecCore now can mean being ready early for the future !

Document Number: P0214R9
Date: 2018-03-16
Reply-to: Matthias Kretz <m.kretz@gsi.de>
Audience: LWG
Target: Parallelism TS 2

DATA-PARALLEL VECTOR TYPES & OPERATIONS

ABSTRACT

This paper describes class templates for portable data-parallel (e.g. SIMD) programming via vector types.

The effort: data & code transformations

- The data transformations:
  - Extracting/filtering the data relevant for the algorithm
  - Fetching the data to VecCore types: AOS -> SOA -> Real_v
- Code transformations:
  - Scalar_t func(Scalar_t input) -> template <typename Real_v> void func(Real_v input, Real_v &output)
  - A = B + C -> A = B + C;
  - A = (B < 0) ? C : D -> A = vecCore::Blend(B < 0, C, D);
  - if (B < 0) A = C -> vecCore::MaskAssign(A, B < 0, C);
  - using namespace vecCore::math

Example: multiple level loops (VecGeom tessellated solid)

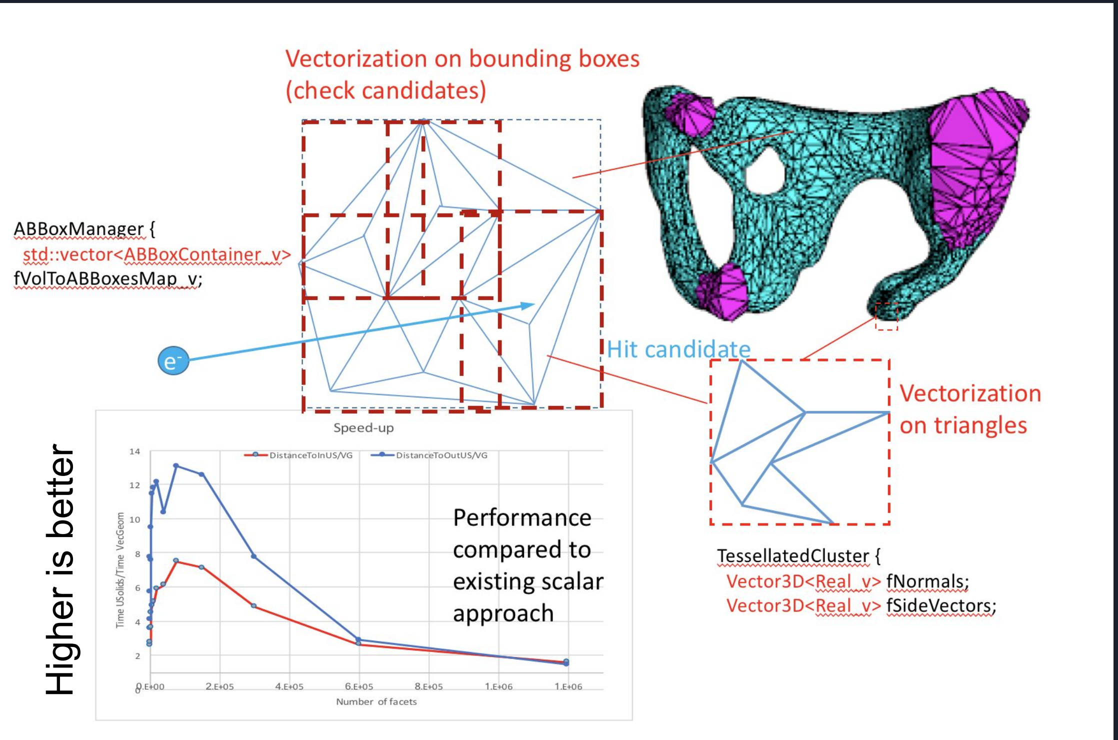
Scalar input: single particle

Solid decomposed in triangles

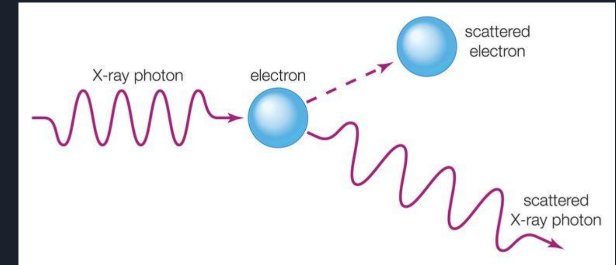
Triangles grouped in clusters having kVectorLength size

Clusters grouped in aligned bounding boxes

Vectorization on multiple levels



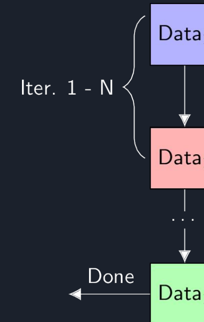
Example: lane refilling (GeantV Compton model)



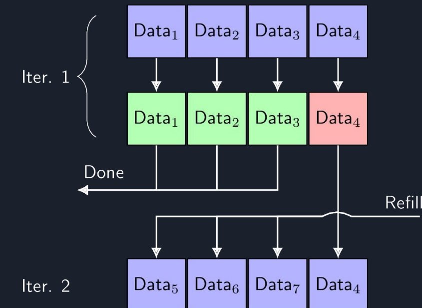
Compton scattering: The fraction of energy transferred to electron has to be sampled for every scattering.

- Alias sampling: x-sections stored as tables of specific data, with one table per gamma energy bin.
  - Vectorizing challenging since data is read from different tables.
- Acceptance/rejection method: a uniform random choice is accepted or rejected to reproduce the PDF.
  - The challenge comes from the conditional recursive nature of the algorithm
  - The strategy used was to refill the "Done" SIMD lanes after each iteration with un-processed input data.
  - Similar approach used for the R-K integration of the equation of motion in fields

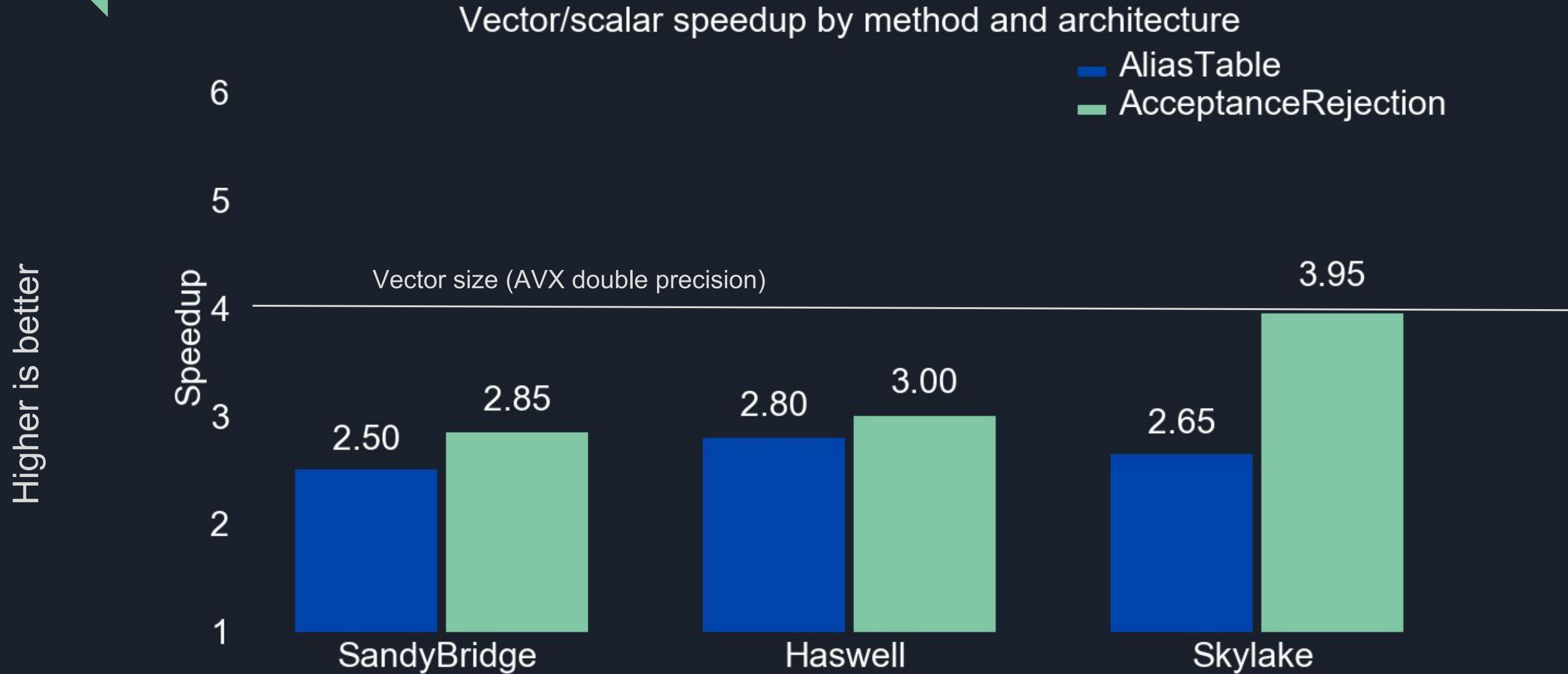
Scalar



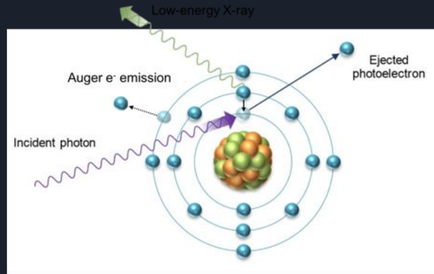
Vectorize over input particles



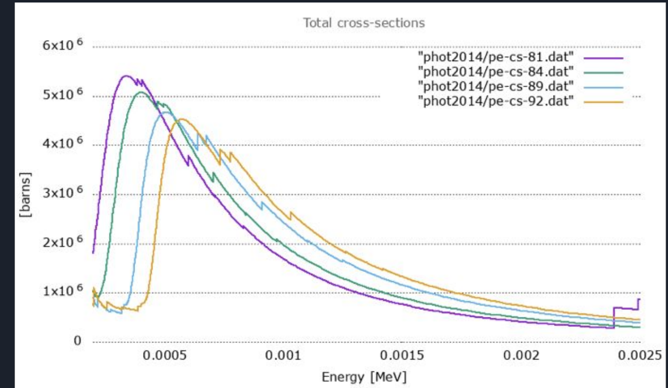
Compton results (AVX, double precision)



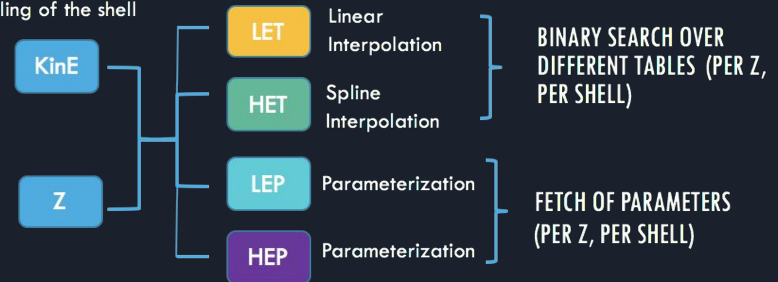
More difficult cases: the PE effect



WIP



- PE total cross-section is not an easy function
  - Fit in two different energy ranges, but not below k-shell binding energy
    - Tabulated cross-sections left for low energies
  - For the final state sampling one need to sample
    - the subshell: This is going through a binary search algorithm (not vectorizable) + linear or spline interpolation
    - the angle: described by the SauterGavrila differential cross-section

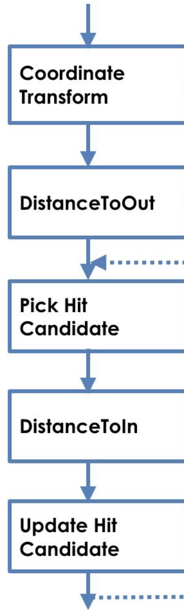


The VecGeom pattern: generic "kernels"

Generic Programming Approach Illustrated

The architectural pattern pioneered in VecGeom

CPU scalar API



instantiate

template kernel

instantiate

template kernel

implemented using

VecCore API/Abstraction

Vc

UMESIMD

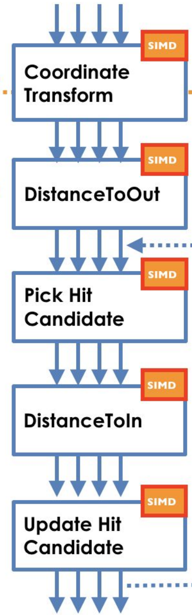
CUDA

Scalar

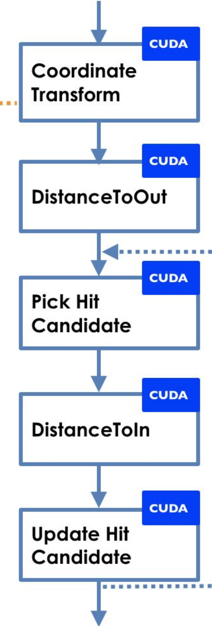
...

compile for different architectures; platforms

CPU SIMD API



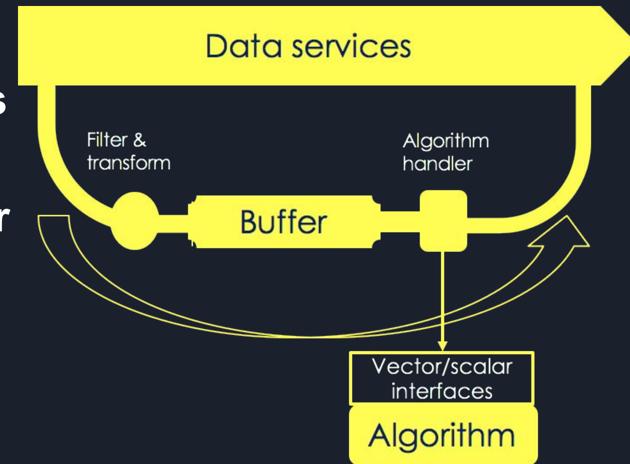
GPU/CUDA API



geant-dev@cern.ch

Bringing the loop inside the algorithm?

- Subscribe to the framework data service & use **filters** integrated in the main event loop
- Use a data transformation service to **extract & buffer** views from the input event data to custom SOA relevant for the algorithm
    - Extract what I need, then reuse many times
- Deliver buffered data to the algorithm using a vector interface
- For which algorithms this may worth doing? How would it work with task-based frameworks? How to deal with parallelism? ...



- Obvious focus shift towards performance-oriented programming in HEP
- Vectorization becoming an important landmark, but:
  - The compilers are not going to solve the problem for us soon…
  - The community (C++) realizing that having vector types and operations support in the language is a must
- VecCore introduces an abstraction level along this line, allowing HEP code to evolve in the right direction and profit ahead of the C++ standard
  - Performance, portability across SIMD architectures, even CPU/GPU
- Path pioneered by VecGeom, but effort needed community-wide (HEP)to build vector-friendly libraries
  - Building also the knowledge-base of “how” and “where” to use this tool
- New approaches on the data and processing flow may boost performance in areas we wouldn’t have thought was possible