**Introduction**

Experiments at CERN have an ever increasing demand for computa-
tional resources, be it for simulated collisions or data analysis. In
the last few decades, this demand has basically been met by hardware
upgrades of the Worldwide LHC Computing Grid. However, with the
new runs of the LHC approaching, the expected increases in beam
luminosity will push this demand far beyond the limits of what fur-
ther hardware upgrades can reach. Therefore, in order to bridge the
widen gap between the needs of the HEP community and the ex-
isting computing resources, HEP software will need to be optimized
to be able to fully exploit SIMD and multithreading parallelism avail-
able in modern hardware.

One of the key areas where performance can be substantially im-
proved in HEP software is SIMD vectorization. Even so, writing effi-
cient SIMD vectorized code is a significant challenge in many large
software projects such as Geant, ROOT, and experiment frameworks.

**The VecCore Library**

The VecCore library was created in order to solve the lack of porta-
lity and unreliable performance problems usually related to SIMD
code by providing a simple API to express SIMD-enabled algo-
rithms that can be dispatched to different backend implementations,
such as SIMD libraries like Vc [https://github.com/edanor/umesimd]
and UME:SIMD [https://github.com/edanor/umesimd] or even CUDA, if
the code has the proper annotations (also supported by VecCore).

Using VecCore, developers can write generic computational kernels
using abstract types that map to the different concrete types in each
backend. The API for vectorization provided by VecCore is architec-
ture agnostic, making it easy to fallback to scalar types on hardware
that does not have SIMD instructions. The API covers the essential
parts of the SIMD instruction set that allows one to write many of
the numerical algorithms needed by HEP software. The main
functions of this API are shown below.

---

**Example: Mandelbrot Set**

In order to demonstrate how an algorithm changes when using the
VecCore API, we created scalar and vectorized implementations of
the calculation of the Mandelbrot set. The vectorized version works
on several pixels at a time, and both algorithms are single-threaded
for simplicity. The final image was generated by coloring a pixel map
depending on the number of iterations before the point di-

---

**Electric Magentic Physics Models**

In GeantV, VecCore has been used to create vectorized versions of
electromagnetic physics models. Since energetic photons and elec-
trons produce particle showers, they are created much more fre-
quently compared to other particle types during simulation.
Therefore, electromagnetic physics processes can take a large portion
of computing time and are the natural first targets for vectorization.
The main electromagnetic processes and vectorization speedups ob-
tained on Intel Xeon Phi™ are shown in the figures below.

**Geometry Algorithms**

The other compute-intensive component of simulations of large LHC-
scale detectors is navigating their geometry. The VecGeom geomety-
library, used in GeantV and recent versions of Geant4, has intro-
duced a vectorized multi particle API based on VecCore to perform
ray casting, distance calculations, and navigation in sets of particles
within detector geometries. The figure below shows performance

**Electromagnetic Processes**

<table>
<thead>
<tr>
<th>Compton Scattering</th>
<th>Pair Production</th>
<th>Photo-Electric Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bremstrahlung</td>
<td>Anihilation</td>
<td>Ionization</td>
</tr>
</tbody>
</table>

---

**Vectorization Speedup of Models on Intel® Xeon Phi™ (KNL)**

<table>
<thead>
<tr>
<th>Number of tracks</th>
<th>Vectorization speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>6.0</td>
</tr>
<tr>
<td>16</td>
<td>4.8</td>
</tr>
<tr>
<td>32</td>
<td>4.0</td>
</tr>
<tr>
<td>64</td>
<td>3.2</td>
</tr>
<tr>
<td>128</td>
<td>2.6</td>
</tr>
<tr>
<td>256</td>
<td>2.2</td>
</tr>
<tr>
<td>512</td>
<td>1.9</td>
</tr>
<tr>
<td>1024</td>
<td>1.7</td>
</tr>
<tr>
<td>2048</td>
<td>1.5</td>
</tr>
</tbody>
</table>

---

**VectorCore Implementation**

VecCore is now integrated into ROOT, where it is being used to vec-
torize classes used in data fitting. For more information, please see
the post “Parrallelization and Vectorization of ROOT Fitting Classes”,
bys Xavier Valls Pla.

---

**Acknowledgements**

VecCore development has been funded by an Intel® Parallel Comput-
ing Center project at São Paulo State University (UNESP), and CERN, as part of the GeantV project. VecCore is now maintained by
the ROOT Team.

---

**References**

1. VecCoreLibrary [https://github.com/root-project/veccore]