Toward a vectorization mechanism in C++

Document number: Draft
Version: 0.1
Date: 2012

Vincent Reverdy (vince.rev@gmail.com)
Laboratory Universe and Theories, Observatory of Paris,
5 place Jules Janssen, 92195 Meudon, France

December 22, 2012

Abstract

This proposal concerns the addition of three helper classes to the standard library in order to provide an easy-to-use vectorization mechanism. The goal is to reduce the current complexity of implementing new containers with optimized vector operations using generic tools based on the CRTP technique. It will also provide a common base for all the constant-size vectors and matrices of graphics and scientific librairies. Finally, this proposal should be able to fill the current lack of vector operations support of the C++ and bring a standardized answer to a lot of technical and scientific basic problems.

Contents

I Motivation **2**

II Impact on the standard **3**

III Design decisions **4**

IV Technical specifications **5**
IV.1 The empty base class `std::vectorizer` 5
IV.2 The constant size vectorization tool `std::static_vectorizer` 5
IV.3 The dynamic size vectorization tool `std::dynamic_vectorizer` 5

V Examples of use **5**

VI References **5**

I Motivation

This proposal comes from a simple observation: an impressive number of colleagues working in a wide variety of engineering and scientific fields are concerned by the lack of standardized tool to design basic vectors and matrices and many of them emphasize the advantage of languages like FORTRAN on this specific point. This fact can be easily confirmed by looking to widely used graphics and scientific libraries and frameworks: oftenly, classes such as `Vector2D`, `Vector3D`, `Tensor3D` or `Matrix4x4` are reimplemented from scratch. Some examples are given in table 1. This results in a huge development effort to design and optimize basic operations on simple things like three-components vectors. So the open question addressed here is : what is the most generic base of all vectorized containers that could be standardized in order to provide an elegant solution to this problem ?

Before going further, it is important to note two key elements. First, even if `std::valarray` provides an optimized mathematical container that supports all basic operations, it is not well designed for composition or inheritance. Consequently, it cannot be used as a common base on which one can add new operators or functions. Second, the problem addressed here is not the same as the one addressed by linear algebra libraries. The goal of the proprosed tool is not to diagonalize 500×500 matrices, or to solve sets of thousands of equations. Consequently, we also avoid the never ending debate of sparse versus non-sparse mathematical containers. To summarize, the goal is to provide a generic mechanism that simplifies and unifies the design of new containers that need optimized vector operations.

This mechanism will reduce the long step of design, implementation and optimization of basic things like a `Vector3D` to the simple inheritance from a `vectorizer` class. A resulting example of use is provided in listing 1. Finally, these tools will have a wide range of use and will allow library designers, software developers, engineers and scientists to implement their own optimized vector containers without having to implement them from scratch.

Listing 1: Basic example of the vectorization syntax

```
1  template <typename T>
2  class MyVector3D
3  : public std::static_vectorizer<T, 3, size_t, MyVector3D>
4  {/* Creates a static size vector */};
5
6  template <typename T, size_t N>
7  class MyVectorN
8  : public std::static_vectorizer<T, N, size_t, MyVectorN, N>
9  {/* Creates a static size vector */};
10
11 template <typename T, size_t N, size_t M>
12 class MyMatrixNxM
13 : public std::static_vectorizer<T, N*M, size_t, MyMatrixNxM, N, M>
14 {/* Creates a static size matrix */};
15
16 template <typename T>
17 class MyVector
18 : public std::dynamic_vectorizer<T, MyVector>
19 {/* Creates a dynamic size vector */};
20
21 template <typename T>
22 class MyMatrix
23 : public std::dynamic_vectorizer<T, MyMatrix>
24 {/* Creates a dynamic size matrix */};
```

Table 1: Examples of basic vectorized containers of some C++ libraries and frameworks.

Library	Category	Example of containers
Qt	Application/GUI Framework	`QVector2D, QVector3D, QVector4D, QGenericMatrix, QMatrix4x4...`
VTK	Visualization	`vtkVector2, vtkVector3, vtkPoints2D, vtkPoints, vtkMatrix3x3, vtkMatrix4x4, vtkTensor...`
OGRE	Game	`SmallVector, Matrix3, Matrix4...`
Irrlicht	Game	`vector2d, vector3d, CMatrix4...`
OpenSceneGraph	Game	`Vec2d, Vec3d, Vec4d, Matrix2, Matrix3...`
Panda3D	Game	`LVector2d, LVector3d, LVector4d, LMatrix3d, LMatrix4d...`
OpenFOAM	Fluid Dynamics	`scalar, vector, tensor...`
Lorene	Astrophysics	`Scalar, Vector, Tensor...`
ROOT	Particle Physics	`DisplacementVector3D, DisplacementVector2D, PositionVector3D, PositionVector2D, Rotation3D, Vector3D, Point3D...`
And many others...	...	...

II Impact on the standard

This proposal is a pure addition to the existing library and consequently it would not affect existing programs. Nevertheless, it requires the addition of a new header shown in table 2.

Table 2: Summary of affected headers

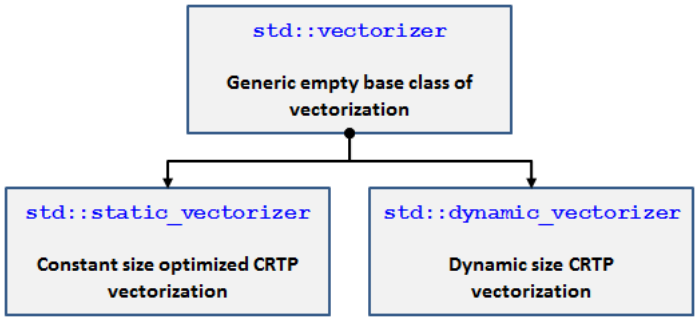
Subclause	Header(s)
IV	`<vectorizer>`

Unless otherwise specified, all components described in this proposal would be declared in namespace `std`. Furthermore, unless otherwise specified, all references to components described in the C++ standard library are assumed to be qualified with `std::`. The design described in part III uses only C++11 features and do not require non-standard extension. The technical specifications described in part IV will require an extensive use of the type traits of the standard library. No current element of the standard library will be modified by or will depend on the tools provided in the `vectorizer` header.

III Design decisions

The design presented in the following is based on original ideas introduced and tested during the development and implementation of the MAGRATHEA framework[1]. The philosophy is quite the same as the one of the `std::iterator` class that provides a generic tool to simplify the creation of new iterators. The goal here is to provide empty classes that simplifies the creation of new vector containers. To do so, the CRTP[2] idiom is used. The resulting tools will allow the user to provide all the standard arithmetic operators, iterator getters, element accessors and some modifier functions to his containers just by inheriting from one of the `vectorizer` classes and overloading some members as the subscript operator. Nevertheless, to keep the design as simple and as generic as possible, no multidimensional operation is provided (as the matrix multiplication) but it will be quite easy for the user to add all the specific operators he wants to his vectorized classes. This ease of modification is a great advantage of the mechanism described in the following paragraphs.

Figure 1: Inheritance relation between the three classes of the `<vectorizer>` header.



The design of the `<vectorizer>` consists of three classes as presented in figure 1:

- `std::vectorizer`: The base class, mainly used for type traits.
- `std::static_vectorizer`: The tool to vectorize constant size containers of the form `Container<typename, IntegralType...>` thanks to CRTP and that allows extended compile-time optimizations.
- `std::dynamic_vectorizer`: The tool to vectorize dynamic size containers of the form `Container<typename>` thanks to CRTP.

These three classes are abstract classes with protected destructors and cannot be used directly. Furthermore, they are empty classes in the sense that they only define methods and no data member.

One limitation of the selected design is that the inherited containers are required to have a certain template form. These template parameters are compatible with standardized containers like `std::vector<typename>` or `std::array<typename, size_t>`. If one want to use another template form, it can inherit a container with the compatible template form, and inherit from it or use alias templates.

[1] The MAGRATHEA (Multi-cpu Adaptive Grid Refinement Analysis for THEoretical Astrophysics) framework is under active development by V. Reverdy and is expected to be released publicly as an open source software in 2013.

[2] The CRTP acronym stands for Curiously Recurring Template Pattern.

IV Technical specifications

All the following is considered as being declared in the `<vectorizer>` header. Unless otherwise specified, all references to components described in the C++ standard library are assumed to be qualified with `std::`.

IV.1 The empty base class `std::vectorizer`

IV.2 The constant size vectorization tool `std::static_vectorizer`

IV.3 The dynamic size vectorization tool `std::dynamic_vectorizer`

V Examples of use

VI References