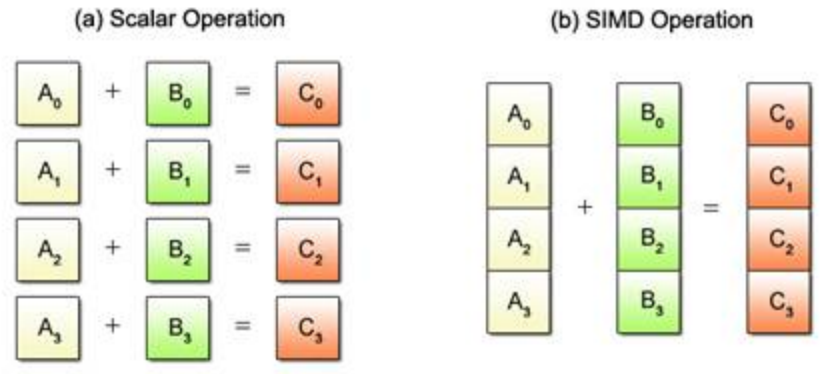
Writing C++ Wrappers for SIMD Intrinsics (1)

Oct 9th, 2014

Introduction

SIMD (and more generally vectorization) is a longstanding topic and a lot has been written about it. But when I had to use it in my own applications, it appeared that most of the articles were theoretical, explaining the principles vectorization lacking practical examples; some of them, however, linked to libraries using vectorization, but extending these libraries for my personal needs was difficult, if not painfull. For this reason, I decided to implement my own library. This series of articles if the result of my work on the matter. I share it there in case someone faces the same problem.

SIMD stands for Single Instruction, Mutiple Data, a class of parallel computers which can perform the same operation on multiple data points simultaneously. Let's consider a summation we want to perform on two sets of four floating point numbers. The difference between scalar and SIMD operations is illustrated below:



Using scalar operation, four add instructions must be executed one after other to obtain the sums, whereas SIMD uses a single instruction to achieve the same result. Thus SIMD operations achieve higher efficiency than scalar operations.

SIMD instructions were first used in the early 1970s, but only became available in consumer-grade chips in the 90s to allow real-time video processing and advanced computer graphics for video games. Each processor manufacturer has implemented its own SIMD instruction set:

- MMX / SSE / AVX (Intel processors)
- 3DNow! (AMD processors)
- Altivec (Motorola processors)
- MDMX (MIPS processors)

Many of these instruction sets still coexist nowadays, and you have to deal with all of them if you want to write portable software. This is a first argument for writing wrappers: capture the abstraction of SIMD operations with nice interfaces, and forget about the implementation you rely on.

Although you can write assembly code to use the SIMD instructions, compilers usually provide built-in functions so you can use SIMD instructions in C without writing a single line of assembly code. These functions (and more generally functions whose implementation is handled specially by the compiler) are called intrinsic functions, often shortened to intrinsics. Of course the SIMD intrinsics depend on the underlying architecture, and may differ from one compiler to other even for a same SIMD instruction set. Fortunately, compilers tend to standardize intrinsics prototype for a given SIMD instruction set, and we only have to handle the differences between the various SIMD instruction sets.

In this series of article, the focus will be on wrapping Intel's SIMD instruction set, although the wrappers will be generic enough so that plugging other instruction sets is easy.

Since SIMD instructions are longstanding, you might wonder if writing your own wrapper is relevant; maybe you could reuse an existing library wrapping these intrinsics. Well, it depends on your needs.

Agner Fog has written some very usefull classes that handle Intel SIMD instruction set (different versions of SSE and AVX), but he doesn't make heavy use of metaprogramming in his implementation. Hence adding a new wrapper (for a new instruction set, a new version of an existing one or even for your own numerical types) requires to type a lot of code that could otherwise have been factorized. Moreover, some essential tools are missing, such as an aligned memory allocator (we will see why you need such a tool later). However his library is a good starting point.

Another library you might want to consider is the Numerical Template Toolbox. Although it has a very comprehensive set of mathematical functions, its major drawback is that it really slows the compilation. Moreover its development and documentation aren't finished yet, and it might be difficult to extend it.

And last but not least, writing your own wrapper will make you confront issues specific to SIMD instructions and make you understand how it works; thus you will be able to use SIMD intrinsics in a really efficient way, regardless of the implementation you choose (your own wrappers or an existing library).

Posted by Johan Mabille Oct 9th, 2014 SIMD, vectorization

Writing C++ Wrappers for SIMD Intrinsics (2) »

Comments