# Using x86/x64 intrinsic functions in C/C++

## Objectives.

1. To write C/C++ programs using Intel SSE intrinsics (assembly coded functions).
2. To understand C/C++ programs using Intel AVX intrinsics (assembly coded functions).

### Aim

The aim of this session is to learn how to write C programs using SSE/AVX x86-64 instrinsics (assembly coded functions)

### Introduction

All modern processors support vectorization. This means that processors have extra hardware components (**wide registers** (up to 512bits) **and wide processing units**) to allow vector processing. Vectorization is the process of processing vectors (multiple values together) instead of single values. Vectorization is also known as Single Instruction Multiple Data (SIMD), as a single instruction is used to process multiple data.

Vectorization dramatically improves the performance of our code. The compilers apply vectorization automatically, this process is called auto-vectorization, but not always with success. Vectorization has nothing to do with the language used.

Vectorization can be applied in several different ways, e.g., a) automatically by the compiler, b) by using C/C++ pragmas, c) using C/C++ instrinsics (this is the method we will be using here), d) directly writing assembly code. By using C/C++ instrinsics we can get high performance (comparable to assembly - assembly code is always faster than any other language) portable and easy to write code.

The code for this week's lab session is located to the 'Vectorization 1st lab session' folder. Download '*code to start*' source files. Open Visual studio and create an empty (C/C++) project. Copy paste the code provided in your project. **For more information about how to create a C/C++ project please see the '*How to Create a C/C++ Project'*** *section* below.

Visual Studio does not support a separate template for C. **Note that C++ is a superset of C**. So, C++ supports everything that comes with C plus extra features such as Object-Oriented Programming, Exception Handling and a rich C++ Library. **So, we will create a C++ project and write C code**.

**The code provided contains multiple files and is advanced. You will not be asked to develop such code. *The aim of this session is just to understand and amend a few routines*.** The code provided, contains two .cpp and one .h source files. It is good practice to put all the declarations in the .h files and the definitions to the .cpp files. The execution of the program always starts from the main() function which is located to the main.cpp file.

### Section 1 – 1st example, add an array with a constant number

This is the 1st code example which adds an array with a constant number and stores the result to another array. This example is explained in the slides too. The files involved to this example are just the *main.cpp*, the *array_constant_addition.cpp* and the *array_constant_addition.h*. *array_constant_addition.h and array_constant_addition.cpp* contain three different implementations of the aforementioned algorithm : a) a normal C routine, b) a routine using SSE instrinsics, c) a routine using AVX instrinsics. You can specify which function to run in main.cpp file under the 't' loop.

In main.cpp file there is code initializes the arrays (initialization_ConstAdd() function), code used to measure the execution time of 't' loop, code for verifying that the ConstAdd_SSE() and ConstAdd_AVX() generate the right output etc.

**Task1**. **Build and run the code.**

By default, the '*Const_Add_SSE()*' routine will run and a message will be printed depending on whether the vectorized version of the algorithm generates the same output as the non-vectorized one. All the functions defined in *array_constant_addition.cpp file,* return the same output, which is the value of 2. This way, '*main()*' knows that we need to compare the output of this algorithm and thus the appropriate if condition will be executed. Inside the if-condition there are two commands. The first one stores a message into an array of strings using '*snprintf()*'. The second, prints a message whether the output is correct or not. To do so, the '*Compare_ConstAdd()*' routine is executed.

***Task2. Measure the execution time***

In this task, we will measure the execution time of *Const_Add_SSE()* routine. **Make you sure you run the project using Ctr+F5 (run without debug);** debugging mode does not provide the actual execution time of the program. **It is important to note that for an accurate measurement, the execution time needs to be at least some seconds.** Thus, given that the execution time of the current routine is lower than 1sec at all times, we run it *'t'* times ('*t*' is a loop variable) and then the overall execution time is divided to the  number of iterations. The upper bound of *'t'* is the 'TIMES_TO_RUN' macro and must be appropriately defined.

Comment the '*Const_Add_SSE()'* routine and uncomment the '*Const_Add_default()'* routine. Build and run the program and measure the execution time of the '*Const_Add_default()'* routine. This routine must be slower. Repeat the experiment and measure the execution time of '*Const_Add_AVX()*' routine. This routine is faster than '*Const_Add_SSE()',* as 8 elements are processed in parallel, not 4.

**Task3. Study the** *'Const_Add_SSE()' and 'Const_Add_AVX()'* routines

Make sure you understand how the *'Const_Add_SSE()' and 'Const_Add_AVX()'* routines work. '*Const_Add_SSE()*' is explained in the slides. *'Const_Add_AVX()'* does the same thing but used AVX technology and thus 8 elements are processed at once. For now, assume that the input size is a multiple of 8. Next week, we will further explain the general solution. All the C instrinisics can be found in https://software.intel.com/sites/landingpage/IntrinsicsGuide/# .

***num2 = _mm_loadu_ps (&v2[i])*** : This command loads 4 v2[] elements from memory and stores them to the 128bit variable num2. These elements are v2[i], v2[i+1], v2[i+2], v2[i+3]. Each array element is of 32bits, so their sum is 128 bits. Keep in mind that num2 is defined as ***' __m128 num2 '***. The input operand of this command must be a memory address (you can check here https://software.intel.com/sites/landingpage/IntrinsicsGuide/#); therefore the ***&*** operator must be used so as the memory address of ***v2[i]*** to be provided.

***_mm_storeu_ps (&v1[i], num3)*** : This command is similar to ***_mm_loadu_ps().*** However, a store is performed and not a load. The contents of the 128bit variable num3 are stored into the memory address ***&v1[i]*** , and therefore the values of v1[i], v1[i+1], v1[i+2] and v1[i+3] are updated.

***num3 = _mm_add_ps (num1, num2)*** : This command will add the packed 4 32bit values of num1 to the packed 4 32-bit values of num2 and put the result into num3.

The instructions above use the SSE technology and process 128bits of data. AVX technology use similar instructions to SSE but processes 256-bit of data. The 256bit registers are defined as ***__m256 ymmm***; . ***_mm_add_ps*** becomes ***_mm256_add_ps*** and ***_mm_load_ps*** becomes ***_mm256_load_ps.***

## Further reading

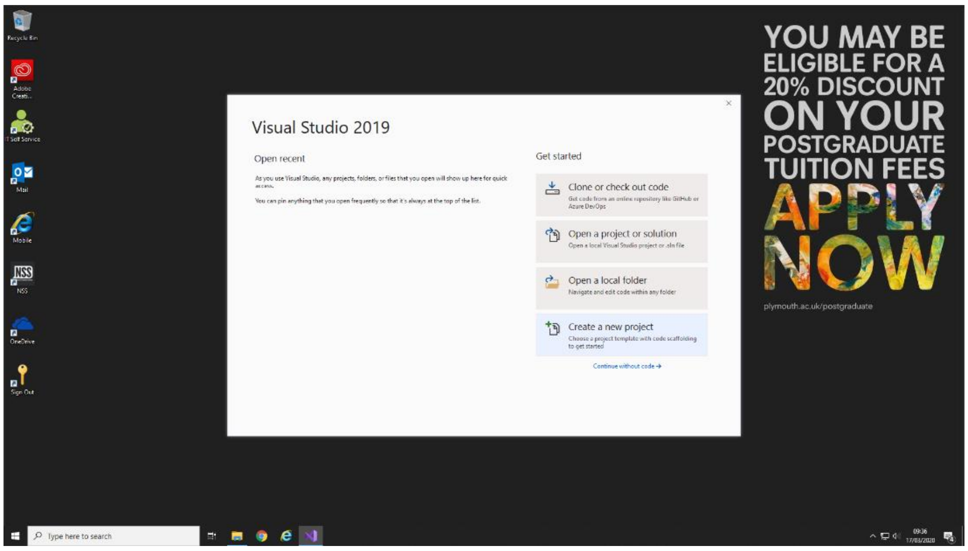
1. *Virtual Workshop (Cornell University), available at [https://cvw.cac.cornell.edu/vector/overview_simd](https://cvw.cac.cornell.edu/vector/overview_simd)*
2. *Tutorial from Virginia University, available at [https://www.cs.virginia.edu/~cr4bd/3330/F2018/simdref.html](https://www.cs.virginia.edu/~cr4bd/3330/F2018/simdref.html)*

## How to Create a C/C++ Project

Visual Studio does not support a separate template for C. **Please note that C++ is a superset of C**. So, C++ supports everything that comes with C plus extra features such as Object-Oriented Programming, Exception Handling and a rich C++ Library. **So, we will create a C++ project and write C code**.

Open Visual studio and select 'create new project' as in Fig.2. Then, type 'C++' in the search area in the top and select the 'empty project' option (Fig.3). Specify the project's name and directory and click 'create' button (Fig.4). Right click on the 'source files' and select 'add'-> 'new item' (Fig.5). Then select 'C++ File (.cpp)' and click 'Add' button (Fig.6). Name the .cpp file appropriately and copy paste the code provided for this file. Repeat this for the other .cpp files too. For the header files (.h), right click on the 'header files' and select 'add'-> 'new item'. Then select 'header File (.h)' and click 'Add' button. Name the .h file appropriately and copy paste the code provided for this file. Repeat this for the other .h files too.

Press F7 to build the code (this option is under the Build tab). Then, press Ctrl+F5 to run the program without debugging or press F5 to run with debugging. To measure the actual execution time of the program you must run the code without debugging (Ctrl+F5).


Visual Studio 2019

Open recent

As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.

You can pin anything that you open frequently so that it's always at the top of the list.

Get started

Clone or check out code
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

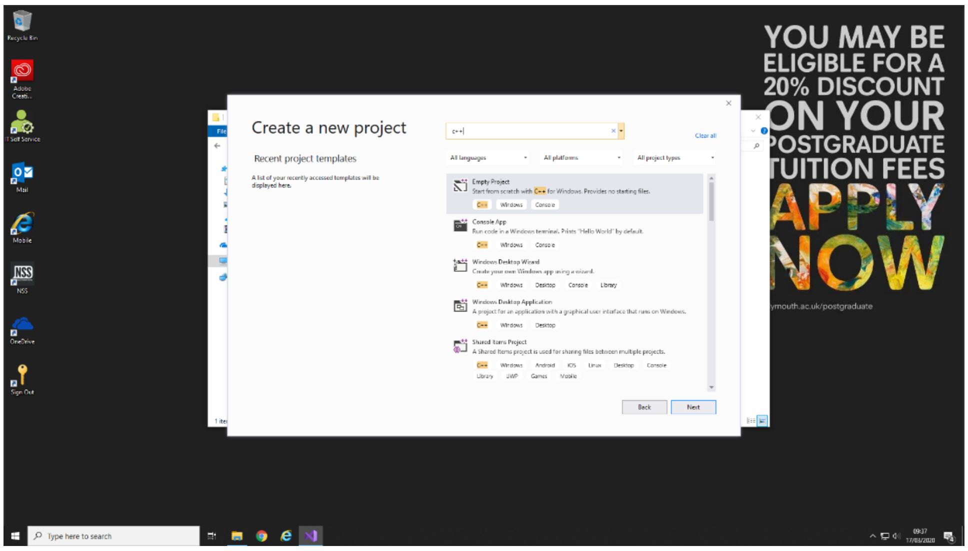
Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

YOU MAY BE ELIGIBLE FOR A 20% DISCOUNT ON YOUR POSTGRADUATE TUITION FEES
APPLY NOW
plymouth.ac.uk/postgraduate
Type here to search

Fig.2. How to create a C++ Project Step1


Create a new project

c++

Recent project templates

A list of your recently accessed templates will be displayed here.

All languages | All platforms | All project types

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console

Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console

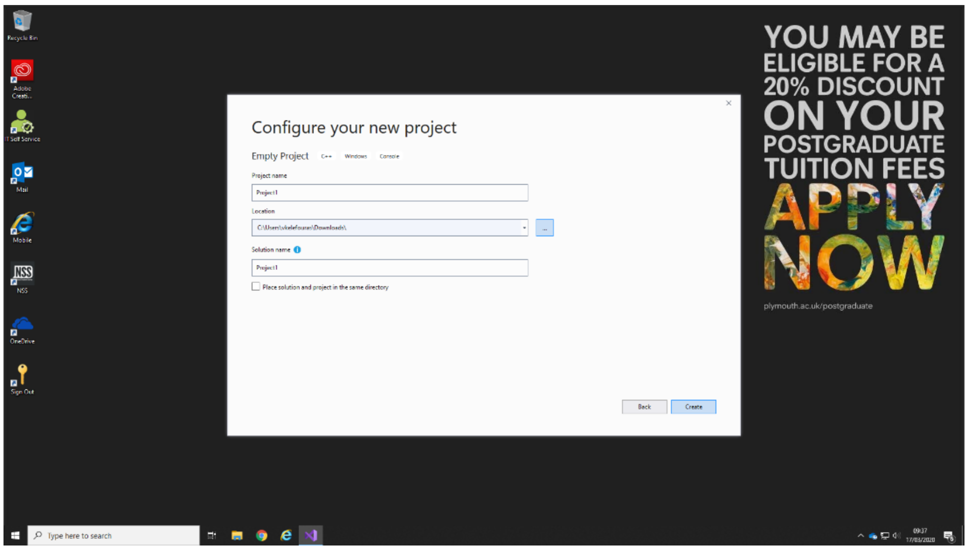
Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library

Windows Desktop Application
A project for an application with a graphical user interface that runs on Windows.
C++ Windows Desktop

Shared Items Project
A Shared Items project is used for sharing files between multiple projects.
C++ Windows Android iOS Linux Desktop Console Library UWP Games Mobile

Clear all
Back
Next
YOU MAY BE ELIGIBLE FOR A 20% DISCOUNT ON YOUR POSTGRADUATE TUITION FEES APPLY NOW
plymouth.ac.uk/postgraduate
Type here to search

Fig.3. How to create a C++ Project Step2


Configure your new project

Empty Project C++ Windows Console

Project name: Project1

Location: C:\Users\vkelefouras\Downloads\

Solution name: Project1

[ ] Place solution and project in the same directory

Back
Create
YOU MAY BE ELIGIBLE FOR A 20% DISCOUNT ON YOUR POSTGRADUATE TUITION FEES APPLY NOW
plymouth.ac.uk/postgraduate
Type here to search

Fig.4. How to create a C++ Project Step3

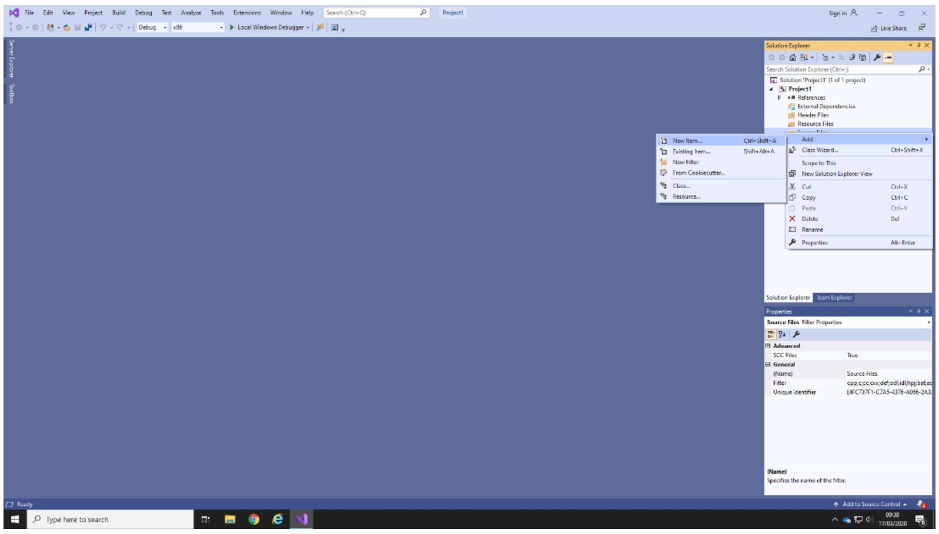
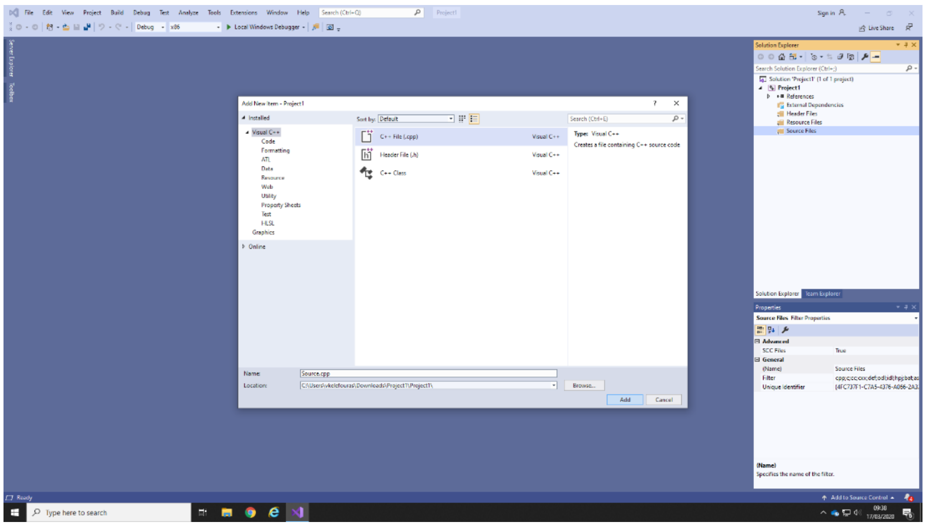


*Fig.5. How to create a C++ Project Step4*



Name: Source.cpp
Location: C:\Users\vkelefouras\Downloads\Project1\Project1\

*Fig.6. How to create a C++ Project Step5*