# 背景知识：

Taf(Trading Application Foundations) provides a framework for low-latency trading system development based on Intel x86-64.

**Features**

* Based on Intel TBB, SIMD(AVX2+), lock-free queue, etc to optimize the performance.
* Nanosecond timestamp available.
* Developed with standard C++, corss-platform (windows and linux) is definitely supported.
* All dependencies included in pack, no need to download any other libs (like TBB).
* Provide C interface for multiple language calls (like Python).
* The delay for a simple application could be less 10us.

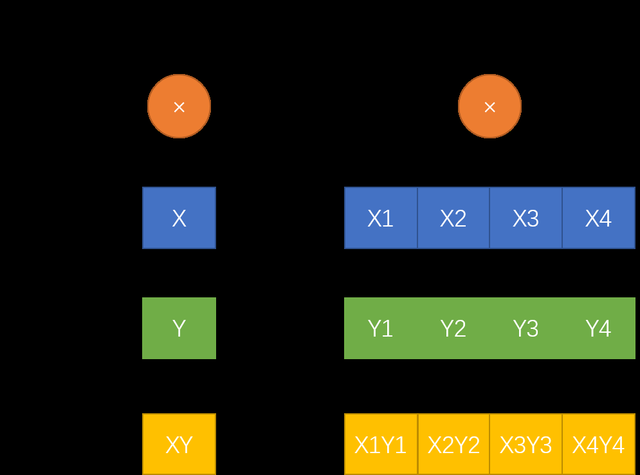
**Before Compilation**

* Check your hardware availability to make sure that AVX2 or AVX512F is supported.
* C++17 compiler is needed.
* TBB libs and binary located in taf/libs/tbb/bin/

# SIMD的演化

要搞明白AVX指令集的作用，首先要讲明白它是什么。定义很简单，它就是x86处理器上面的一套SIMD指令集，是经典的SSE系列指令集的直接继承者。那么SIMD又是什么呢？

单指令流多数据流操作（**S**ingle **I**nstruction Stream, **M**ultiple **D**ata Stream）的思路就被引入了，它让输入一次指令就操作多组数据变成了可能。



MMX指令集->SSE->AVX

AVX指令集带来了巨大的革新，其中最为主要的是，它在兼容SSE指令集性的同时，将SSE时代最大宽度为128-bit的寄存器拓宽到了256-bit。

宽度越大，处理器的计算能力也就越强，尤其是在浮点运算方面，理论上提升有一倍之多，而实际应用中，如果优化得当，其提升幅度还要大一些。但是，新指令集在带来性能增长的同时也带来了另一个让人感到头痛的问题——功耗。

Intel官方也在2014年的一份AVX指令集优化白皮书中明确说明使用AVX指令集需要额外的电压和电流。

# SIMD的应用场景

在Intel的推广之下，现如今已经有大量的生产力应用支持它了，主要在渲染、视频编码、加解密和数学计算等方面有应用，新的AVX-512还针对深度学习推出了AVX-512 VNNI子集。

渲染方面最常见的有Blender，它不仅仅在我们的测试中被用的多，是真的有很多人都会用它做动画或者CG图，它的渲染引擎可以调用AVX2指令集进行加速计算，吃满你的CPU。

跟渲染方面有点搭边的就是视频编码了，x264和x265这两个知名开源视频编码器想必已经不用再多介绍了，它们都在前几年中纷纷加入了对于AVX指令集的支持，后者甚至加入了针对AVX-512的支持，不过还需要继续优化。

深度学习方面，Google著名的开源深度学习框架Tensorflow在1.6版本之后就已经需要一颗支持AVX指令集的CPU了，换言之，它应用了AVX指令集。

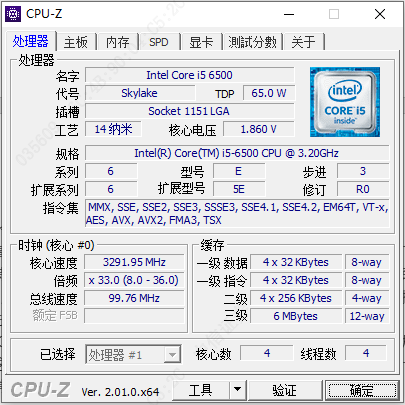
加解密计算场景中对CPU的计算吞吐量有较大的要求，此时AVX指令集就可以发挥作用，常见的软件支持就有OpenSSL这个堪称是互联网基石的加密库，另外像很多程序会使用的libsodium加密库也提供了从AVX到AVX-512的优化，而Linux内核也支持使用AVX和AVX2指令集进行加解密计算，还会配合AES-NI这个专用的指令集。

一般在游戏中CPU负责除了图形以外的杂活，比如说计算各种NPC的运动路径，计算各种动体的轨迹这样的杂活。不过近两年也有厂商想让Intel参与进游戏图形计算。

AVX的前景非常广阔。

# 判断机器是否支持SSE、AVX、AVX-512

## **Windows下直接用软件检查**，比如CPU-Z这类软件



## Linux下用命令行判断

[gcc - How to detect SSE/SSE2/AVX/AVX2/AVX-512/AVX-128-FMA/KCVI availability at compile-time? - Stack Overflow](https://stackoverflow.com/questions/28939652/how-to-detect-sse-sse2-avx-avx2-avx-512-avx-128-fma-kcvi-availability-at-compile)

用下面的哪个命令取决于想用哪个指令集

### $ gcc -msse3 -dM -E - < /dev/null | egrep "SSE|AVX" | sort

得到：

#define \_\_SSE\_\_ 1

#define \_\_SSE2\_\_ 1

#define \_\_SSE2\_MATH\_\_ 1

#define \_\_SSE3\_\_ 1

#define \_\_SSE\_MATH\_\_ 1

或者

### $ gcc -mavx2 -dM -E - < /dev/null | egrep "SSE|AVX" | sort

得到

#define \_\_AVX\_\_ 1

#define \_\_AVX2\_\_ 1

#define \_\_SSE\_\_ 1

#define \_\_SSE2\_\_ 1

#define \_\_SSE2\_MATH\_\_ 1

#define \_\_SSE3\_\_ 1

#define \_\_SSE4\_1\_\_ 1

#define \_\_SSE4\_2\_\_ 1

#define \_\_SSE\_MATH\_\_ 1

#define \_\_SSSE3\_\_ 1

或者

### $ gcc -march=knl -dM -E - < /dev/null | egrep "SSE|AVX" | sort

得到

#define \_\_AVX\_\_ 1

#define \_\_AVX2\_\_ 1

#define \_\_AVX512CD\_\_ 1

#define \_\_AVX512ER\_\_ 1

#define \_\_AVX512F\_\_ 1

#define \_\_AVX512PF\_\_ 1

#define \_\_SSE\_\_ 1

#define \_\_SSE2\_\_ 1

#define \_\_SSE2\_MATH\_\_ 1

#define \_\_SSE3\_\_ 1

#define \_\_SSE4\_1\_\_ 1

#define \_\_SSE4\_2\_\_ 1

#define \_\_SSE\_MATH\_\_ 1

#define \_\_SSSE3\_\_ 1

或者

### $ gcc -march=skylake-avx512 -dM -E - < /dev/null | egrep "SSE|AVX" | sort

得到

#define \_\_AVX\_\_ 1

#define \_\_AVX2\_\_ 1

#define \_\_AVX512BW\_\_ 1

#define \_\_AVX512CD\_\_ 1

#define \_\_AVX512DQ\_\_ 1

#define \_\_AVX512F\_\_ 1

#define \_\_AVX512VL\_\_ 1

#define \_\_SSE\_\_ 1

#define \_\_SSE2\_\_ 1

#define \_\_SSE2\_MATH\_\_ 1

#define \_\_SSE3\_\_ 1

#define \_\_SSE4\_1\_\_ 1

#define \_\_SSE4\_2\_\_ 1

#define \_\_SSE\_MATH\_\_ 1

#define \_\_SSSE3\_\_ 1