# SIMD

**Single instruction, multiple data (SIMD)**是并行计算机的一类(按照Flynn分类法)。它描述了具有多个处理元素(multiple processing elements)的计算机，可以在多个数据点(data points)上同时(simultaneously)完成相同的操作。这种机器利用数据级并行(data level parallelism)(不是并发, concurrency): 在某一时刻，只有一个指令，但有多个同时发生的(simultaneous)计算(computations)，也即并行计算。SIMD特别适用于一些常见的任务，例如：调整数字图像的对比度，或者调整数字音频的音量。大部分现代CPU设计都包含了SIMD指令，来提高多媒体使用的性能。不要把SIMD和SIMT搞混了，前者在data level工作，后者要利用线程。

## 优点

一个利用SIMD的应用程序(application)可以在大量数据点上加上或减去相同的值(应该是一个指令完成的吧)， 这是许多多媒体应用的常见操作。举例，修改一个图像的亮度。图像的每个像素由三个值组成，分别表示颜色的红色(R)、绿色(G)和蓝色(B)部分的亮度。为了改变亮度，从内存中(memory)读取R, G和B值，然后从R, G, B中减去或加上一个值(a value)，最后得到的结果值(values)被写回内存。

使用SIMD processor，这个过程会有两个改进。首先，数据是按块理解的，一次可以加载多个值。和普通的一系列指令“取回这个像素，现在再取回下一个像素”不同，SIMD processor会发出单个指令“取回n个像素” (此处n是一个数值，由CPU的设计决定)。由于各种因素，SIMD这种一个指令取回多个像素的方式比传统CPU设计中一个一个地取回像素需要的时间少很多。

另外一个优点时，这个指令会在单个操作中(a single operation)操作(operate)所有加载的数据。换句话说，假如一个SIMD系统一次加载8个数据点，则一个`add`操作(比如，使值增加1)，会同时作用在8个数上，即这8个数同时加1。

## 缺点

- 不是所有的算法都可以容易地被向量化。即一些程序不太容易使用SIMD。
- 需要容量大的寄存器，增加计算消耗、需要求一定的芯片区域(可理解为芯片的数量/面积是一定的，用于作缓存/寄存器的部分多了，则用于计算的部分就少了，总的计算消耗的时间会变长)。
- 当前，实现一个使用SIMD指令的算法通常要求人工参与，例如：大部分编译器不会为典型的C程序生成SIMD指令。在编译器中自动向量化(Automatic vectorization)是一个很活跃的计算机研究方向。
- 使用某些SIMD指令集编程可能会涉及到许多低层挑战( low-level challenges):
    - SIMD可能在数据对齐上有限制; 只熟悉某种架构的程序员可能不希望这样的事发生。
    - 获取数据到SIMD寄存器中并把它们分散到正确的目标位置是一个棘手的(tricky )问题，可能需要一些交换操作(permute operations)，并且是低效的。

- 某些特殊的指令，如：旋转或三个操作数的加法，在某些SIMD指令集中是不可用的。
- 指令集是架构相关的(architecture-specific): 某些处理器完全缺少SIMD指令，则程序员必须提供非向量化的实现。
- 不同的架构具有不同的寄存器的大小(如:64, 128, 256, 512 bits)和指令集，这意味着程序员必须提供多种向量化实现来在不同的CPU上运行，以便得到最优的效率。

References:

- SIMD