# Go-On-ARM SIMD优化入门案例

## 1 Golang开发环境准备

环境准备

工具介绍

perf

Benchstat

## 2参与社区方法

包含签署贡献协议，下载源码、运行别人CL、自己提交等完整过程

参考一下资料写

<https://stackoverflow.com/questions/25716920/what-does-cl-mean-in-a-commit-message-what-does-it-stand-for>

<https://golang.org/doc/contribute.html>

## 3 使用SIMD优化

场景介绍

SIMD技术全称Single Instruction Multiple Data，即单指令多数据流，通过单条指令并行操作一组数据替换原来的多条指令或循环操作，实现性能提升。ARM64支持的SIMD指令数约400个左右，包含数据加载和存储、数据处理、压缩、加解密等。ARM64包含32个SIMD向量寄存器用于SIMD操作，可以批量加载一组数据到向量寄存器中，使用SIMD指令对向量寄存器中的数据运算后，批量存到内存。SIMD技术常用于多媒体、数学库、加解密算法库等包含循环处理数组元素的场景，通过SIMD指令和向量寄存器的帮助减少其中数据加载和存储、数学运算、逻辑运算、移位等常用操作所需的指令条数。

3.1分析

我们通过Golang官方的一个SIMD优化案例来进行介绍，该CL地址为：

<https://go-review.googlesource.com/c/go/+/71110>

优化前的代码使用Golang汇编编写，实现在src/runtime/asm\_arm64.s中，如下所示：

// memequal(p, q unsafe.Pointer, size uintptr) bool  
TEXT memequal(SB),NOSPLIT,$-8-25  
 MOVD a+0(FP), R1  
 MOVD b+8(FP), R2  
 MOVD size+16(FP), R3  
 ADD R1, R3, R6  
 MOVD $1, R0  
 MOVB R0, ret+24(FP)  
 CMP R1, R2  
 BEQ done  
loop:  
 CMP R1, R6  
 BEQ done  
 MOVBU.P 1(R1), R4  
 MOVBU.P 1(R2), R5  
 CMP R4, R5  
 BEQ loop  
  
 MOVB $0, ret+24(FP)  
done:  
 **RET**

该函数的定义在src/runtime下面的stubs.go文件中，如下所示：

func memequal(a, b unsafe.Pointer, size uintptr) bool

参数a, b的类型实际是两个int指针，表示两个数组，参数size表示数组的大小，简单来说该函数就是按顺序挨个比较两个数组中的元素是否相等。

优化前代码逻辑简析见下图：



根据上图代码解析可以看到，该函数是循环操作两个数组中的元素，当前的处理方式每次循环只能取1 byte进行比较，即每byte数据要耗费两个读取操作和两个比较操作，性能不高。

3.2优化

3.3验证

要求图文结合，图片为主，可不借助其他材料上手完成整个操作