芯来科技
NUCLEI

AI关键算子RVV性能优化

舒卓
Nuclei Technology

Contents:

芯来科技
NUCLEI

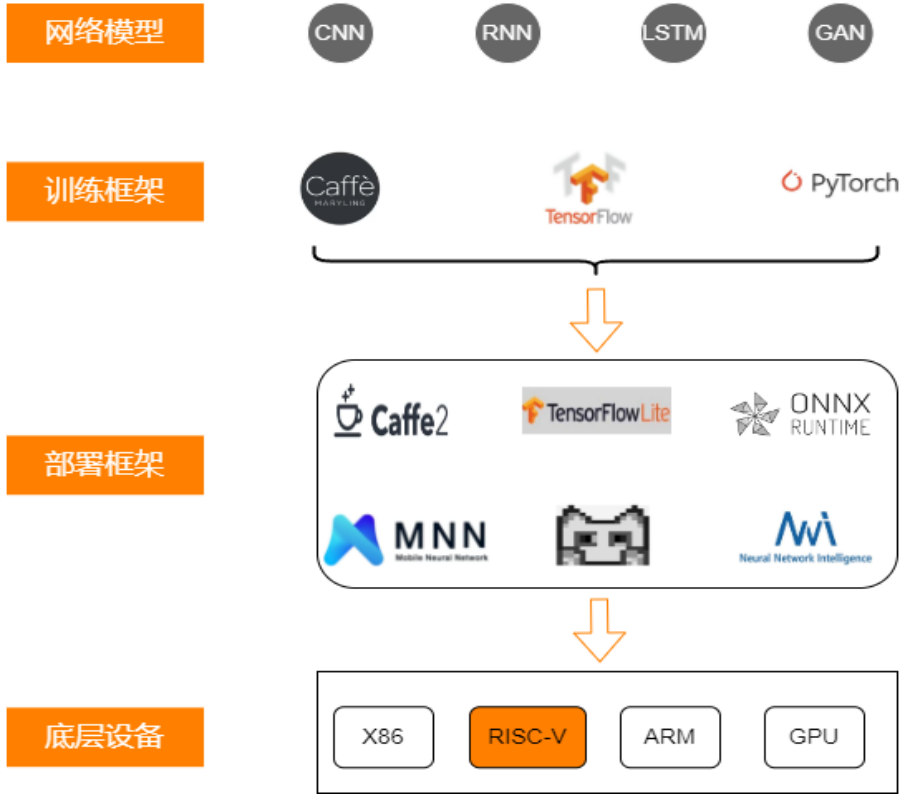
- 嵌入式 AI 框架
- Nuclei RISC-V V 扩展简介
- 使用 V 扩展优化 AI 关键算子示例
- 在 Nuclei Evalsoc 上实测的提升效果

嵌入式 AI 框架

芯来科技
NUCLEI

嵌入式 AI 框架 指专门为资源受限的嵌入式设备设计的机器学习框架。

由于嵌入式设备资源有限，常见做法是采用 **训练-推理分离**，即在服务器(多核CPU/GPU)上训练模型，然后在嵌入式设备执行模型推理。

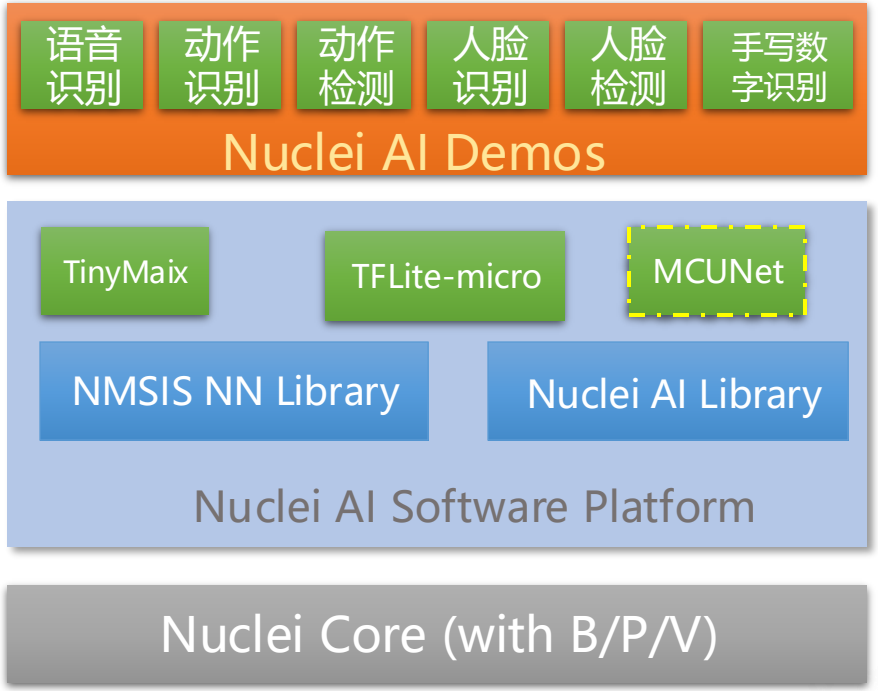


Nuclei 嵌入式 AI 框架

芯来科技 NUCLEI

Nuclei 基于 tflm/tinymaix 等推理框架进行了适配，提供 NMSIS NN Library 以及 Nuclei AI Library 等库，用户可以轻松的获得RISC-V P/V扩展加速能力。

- **Nuclei NN Library**：基于 CMSIS-NN 进行深度 P/V 扩展优化
  链接：https://github.com/Nuclei-Software/NMSIS
- **Nuclei AI Library**: 使用 V 扩展优化常见的AI算子
  链接：https://github.com/Nuclei-Software/nuclei-ai-library
- **Nuclei AI demo**：基于 Nuclei SDK 适配，上手简单，并提供丰富示例
  链接：https://github.com/Nuclei-Software/npk-tflm
  https://github.com/Nuclei-Software/npk-tinymaix

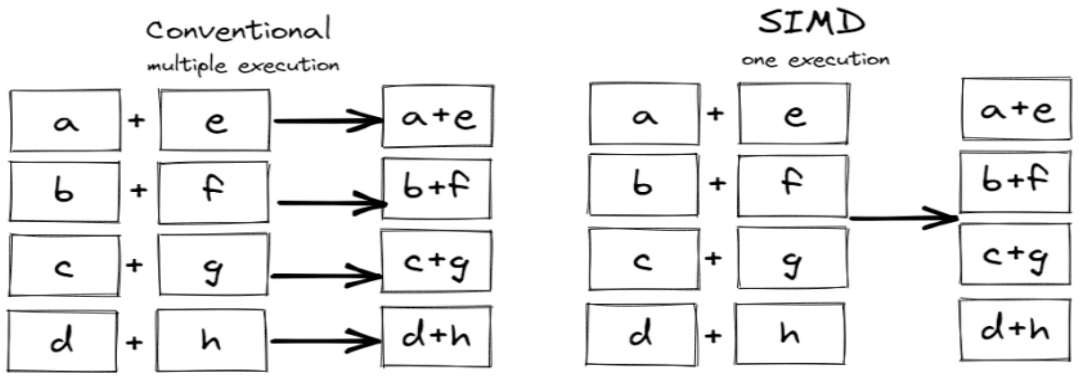


Nuclei RISC-V V 扩展

芯来科技
NUCLEI

Nuclei V 扩展特性:

- 支持 RVV 1.0
- VLEN 长度（128/512/1024bit）可配
- 支持整数/定点/浮点运算
- 支持最新 RVV intrinsic(v0.12.0)
- 用户可以使用 Nuclei 的 NICE 硬件扩展接口，添加自定义 Vector 指令

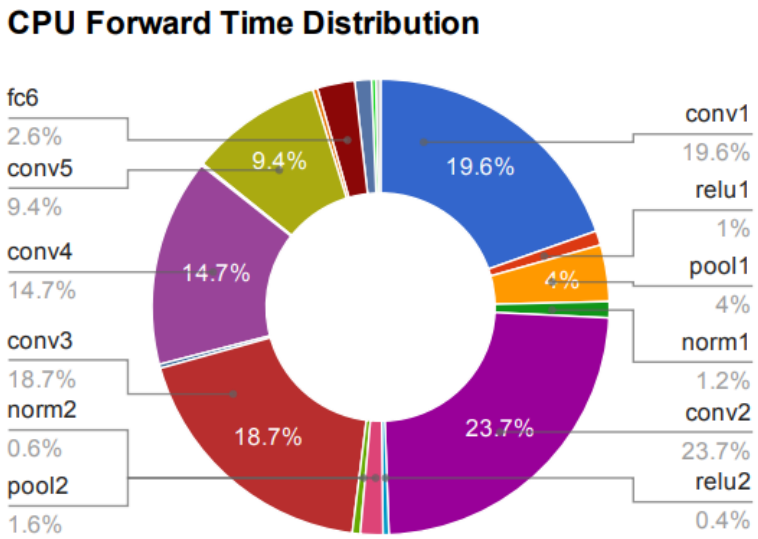


嵌入式AI的几个关键算子- CNN网络

芯来科技 NUCLEI

CNN 网络典型的层：卷积层、激活层、池化层、全连接层等

下图为 Alexnet 网络在 CPU 进行推理的 benchmark，由图可以看出，卷积层和全连接层占用了 89% 的时间，**如何高效地进行卷积层和全连接层的计算是提升深度学习推理性能的关键点。**



图片来源于 Jia, Yangqing.《Learning semantic image representations at a large scale. University of California, Berkeley》

使用 V 扩展优化-GEMM

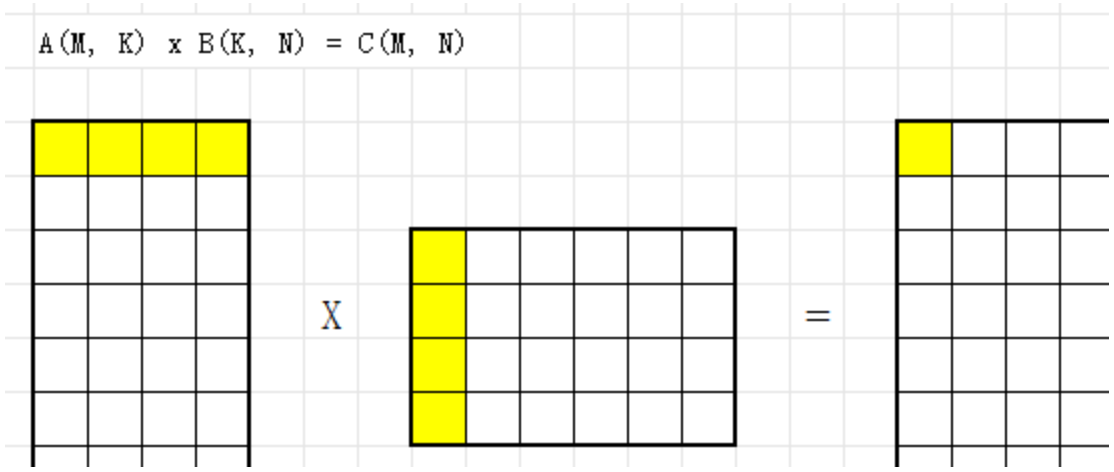
芯来科技 NUCLEI

GEMM算子：即通用矩阵乘， GEMM 是非常重要的算子。

对于CONV2D 常用的优化方法是 im2col + GEMM 优化，而全连接也是特殊的 GEMM。

使用 RVV 优化 GEMM 有如下几种方法：

- **方法1**：直接使用 Reduction 指令




```
void sgemm_opt_rvv_redsum(int32_t m, int32_t n, int32
                float *a, int32_t lda,
                float *b, int ldb,
                float *c, int ldc)
{
    float sum;

    for (int32_t ii = 0; ii < m; ++ii)
    {
        for (int32_t jj = 0; jj < n; ++jj)
        {
            float *pInA = a + ii * k;           /* Inp
            float *pInB = b + jj;               /* Inp
            size_t blkCnt = k;
            size_t l;
            ptrdiff_t bstride = 4;               // 32
            vfloat32m8_t v_inA, v_inB;
            vfloat32m8_t vmul;
            vfloat32m1_t vsum;
            l = __riscv_vsetvl_e32m1(1);
            vsum = __riscv_vfmv_s_f_f32m1(0.0f, l);
            for (; (l = __riscv_vsetvl_e32m8(blkCnt)
```

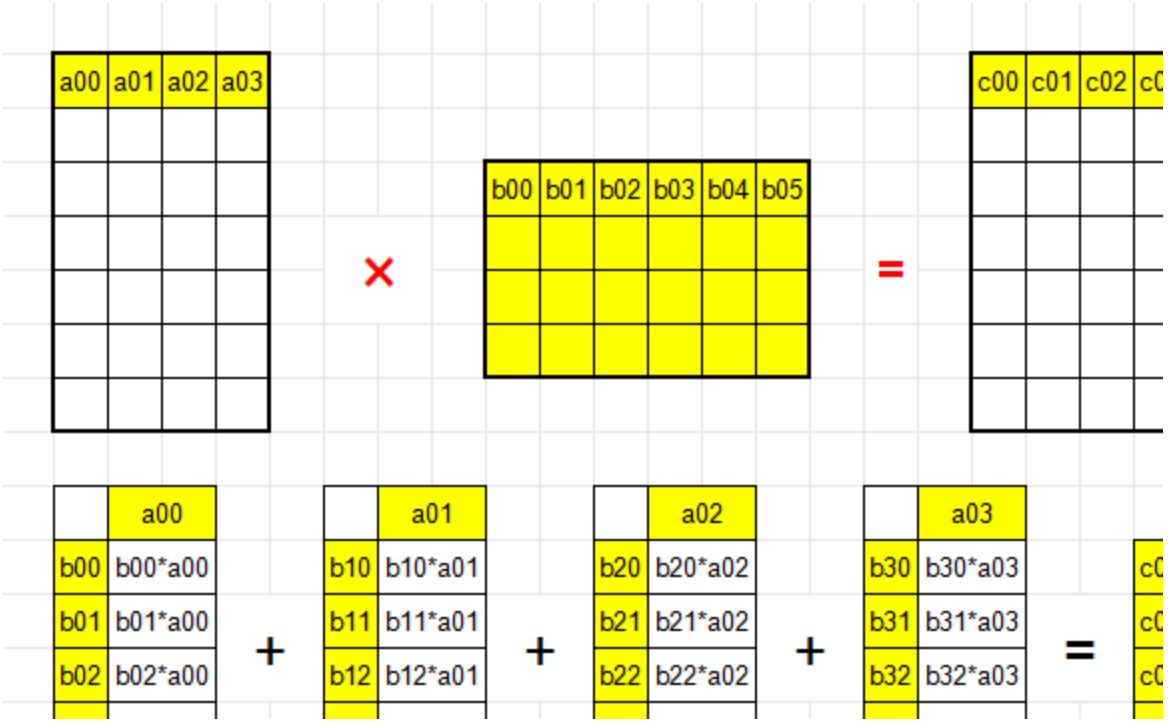
使用 V 扩展优化-GEMM

芯来科技 NUCLEI

- **方法2**：使用 macc 指令，一次可以算多个点

方法2有几个优点：

1. Reduction 指令效率较低，应尽量避免使用
2. 充分”榨取”已经load的数据，减少 load 操作




```
/* ch = 4, mul = 4 */
for (jj = m / 4; jj > 0; jj--) {
    px = c;
    pInB = b;

    for (ii = n; ii > 0; ii -= l) {
        l = __riscv_vsetvl_e32m4(ii);

        pInA = a;

        vres0m4 = __riscv_vfmv_v_f_f32m4(0.0, l);
        vres1m4 = __riscv_vmv_v_v_f32m4(vres0m4, l);
        vres2m4 = __riscv_vmv_v_v_f32m4(vres0m4, l);
        vres3m4 = __riscv_vmv_v_v_f32m4(vres0m4, l);
        for (kk = 0; kk < k; kk++) {
            va0m4 = __riscv_vle32_v_f32m4(pInB + kk * ldb,
            vres0m4 = __riscv_vfmacc_vf_f32m4(vres0m4, *(pI
            vres1m4 = __riscv_vfmacc_vf_f32m4(vres1m4, *(pI
            vres2m4 = __riscv_vfmacc_vf_f32m4(vres2m4, *(pI
            vres3m4 = __riscv_vfmacc_vf_f32m4(vres3m4, *(pI
            pInA++;
        }
        __riscv_vse32_v_f32m4(px, vres0m4, l);
        __riscv_vse32_v_f32m4(px + ldc, vres1m4, l);
```

使用 V 扩展优化-Exp

芯来科技 NUCLEI

Softmax/Elu/Silu 等激活函数调用 Exp 函数，直接使用 math 库中的 Exp 函数效率较低，所以可以考虑使用RVV优化。

- **方法1**：直接基于泰勒展开进行 RVV 优化可能收敛速度很慢(除非x在0附近)，需要考虑其它方法

$$e^x = \sum_{n=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

- **方法2**：泰勒展开 + IEEE754 浮点标准

$$e^x = 2^{x \log_2(e)} = 2^{x_i + x_f}$$

$$x_i = \text{int}(x \log_2(e))$$
$$x_f = x \log_2(e) - x_i$$
$$2^{x_i} = 2^{23}(x_i + 127)$$
$$2^{x_f} = 1 + \frac{\ln(2)x_f}{1!} + \frac{(\ln(2)x_f)^2}{2!} + \frac{(\ln(2)x_f)^3}{3!} + \cdots$$

使用 V 扩展优化-Exp

芯来科技 NUCLEI

Exp 的 RVV 实现如下：

有如下几个技巧：

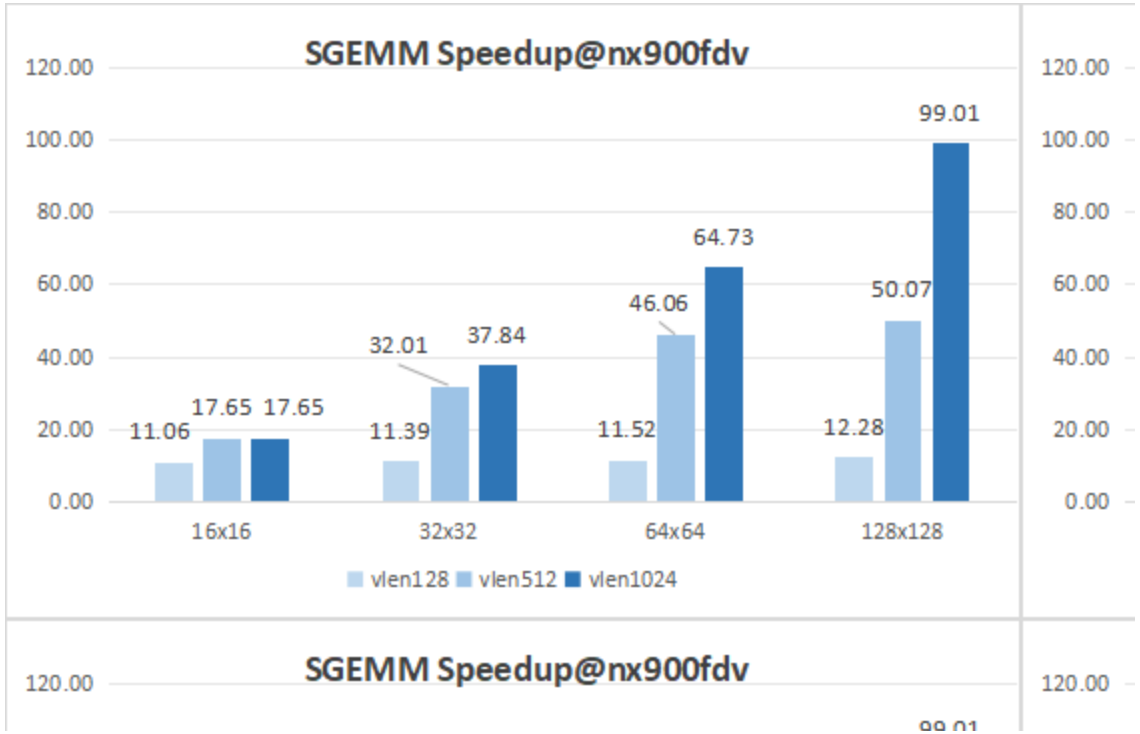
1. 尽量减少 vfdiv 指令
2. lmul 尽量取最大8 （在 V 寄存器够用的情况下）

```
void exp_f32_v(float32_t *src, float32_t *dest, int32_t len)
{
  uint32_t vblkCnt = len;                                   /* Loop counter */
  size_t l;
  vfloat32m8_t vx, vy, vz;
  vint32m8_t vx_int;

  for (; (l = vsetvl_e32m8(vblkCnt)) > 0; vblkCnt -= l) {
    vx = vle32_v_f32m8(src, l);
    src += l;
    vx = vfmul_vf_f32m8(vx, 1.4426950408889634f, l);              // log2(e)
    vx_int = vfcvt_rtz_x_f_v_i32m8 (vx, l);
    vx = vfsub_vv_f32m8(vx, vfcvt_f_x_v_f32m8(vx_int, l), l);

    vx_int = vadd_vx_i32m8(vx_int, 127, l);
    vx_int = vmul_vx_i32m8(vx_int, (1 << 23), l);
    vy = vreinterpret_v_i32m8_f32m8 (vx_int);

    vx = vfmul_vf_f32m8(vx, 0.693147180559945f, l);                 // ln2
    vz = vfmul_vf_f32m8(vx, 1.0 / 5040, l);                         // 1/7!
    vz = vfmul_vv_f32m8(vx, vfadd_vf_f32m8(vz, 1.0 / 720, l), l);   // 1/6!
    vz = vfmul_vv_f32m8(vx, vfadd_vf_f32m8(vz, 1.0 / 120, l), l);   // 1/5!
    vz = vfmul_vv_f32m8(vx, vfadd_vf_f32m8(vz, 1.0 / 24, l), l);    // 1/4!
    vz = vfmul_vv_f32m8(vx, vfadd_vf_f32m8(vz, 1.0 / 6, l), l);     // 1/3!
    vz = vfmul_vv_f32m8(vx, vfadd_vf_f32m8(vz, 1.0 / 2, l), l);     // 1/2!
    vz = vfmul_vv_f32m8(vx, vfadd_vf_f32m8(vz, 1.0, l), l);         // 1/1!
    vz = vfadd_vf_f32m8(vz, 1, l);

    vy = vfmul_vv_f32m8(vy, vz, l);
    vse32_v_f32m8(dest, vy, l);
    dest += l;
  }
}
```

在Nuclei Evalsoc上实测的提升效果-GEMM

芯来科技 NUCLEI

下图是在 Nuclei nx900fdv 上实测 GEMM 算子 RVV 优化提升倍数：



f32

vpu VLEN=128 DLEN=128	16x16	32x32	64x64
sgemm_c	14147	102943	788480
sgemm_v	1261	8958	68099
sgemm_c/sgemm_v	11.22	11.49	11.58

vpu VLEN=512 DLEN=512	16x16	32x32	64x64
sgemm_c	13875	101900	783873
sgemm_v	786	3183	17018
sgemm_c/sgemm_v	17.65	32.01	46.06

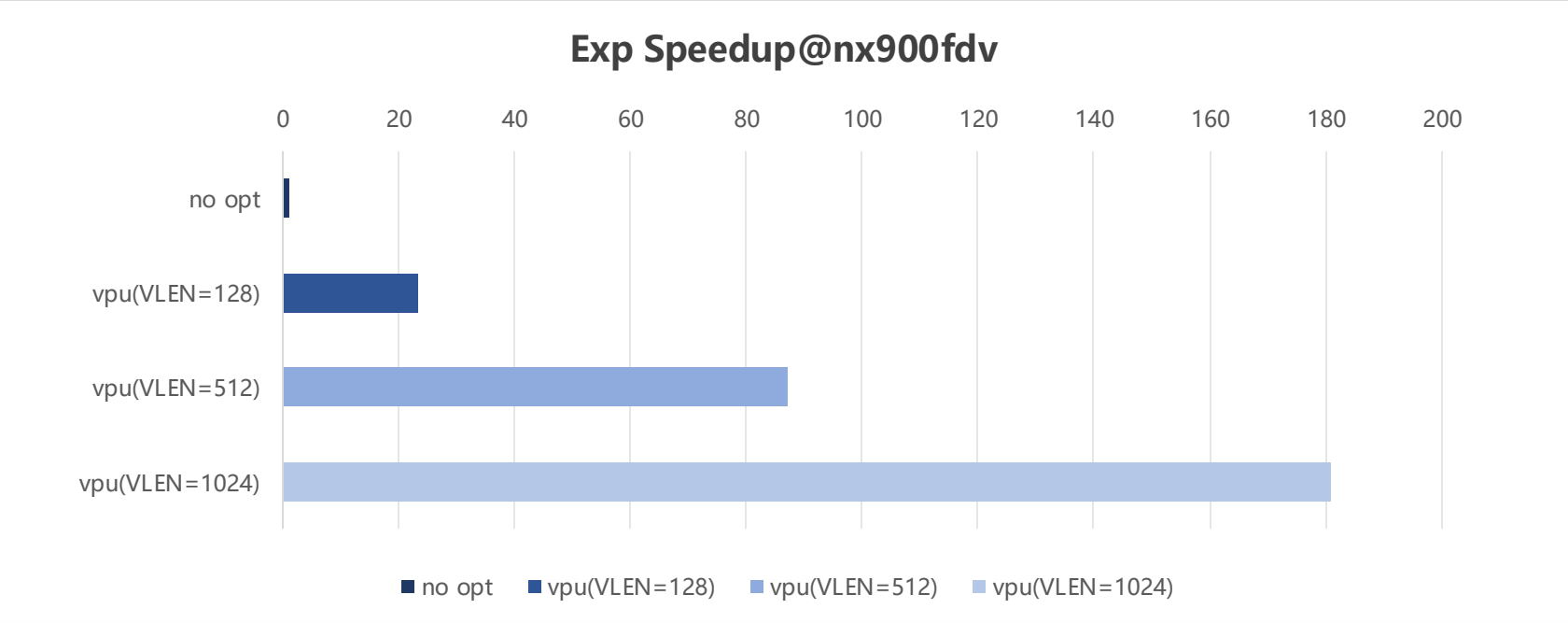
vpu VLEN=1024 DLEN=1024	16x16	32x32	64x64

Note: 16x16 表示 A(16, 16) x B(16, 16) = C(16, 16)，等等

在Nuclei Evalsoc上实测的提升效果-Exp

芯来科技 NUCLEI

下图是在 Nuclei nx900fdv 上实测 Exp 算子 RVV 优化的提升倍数：



芯来科技
NUCLEI

THANK YOU