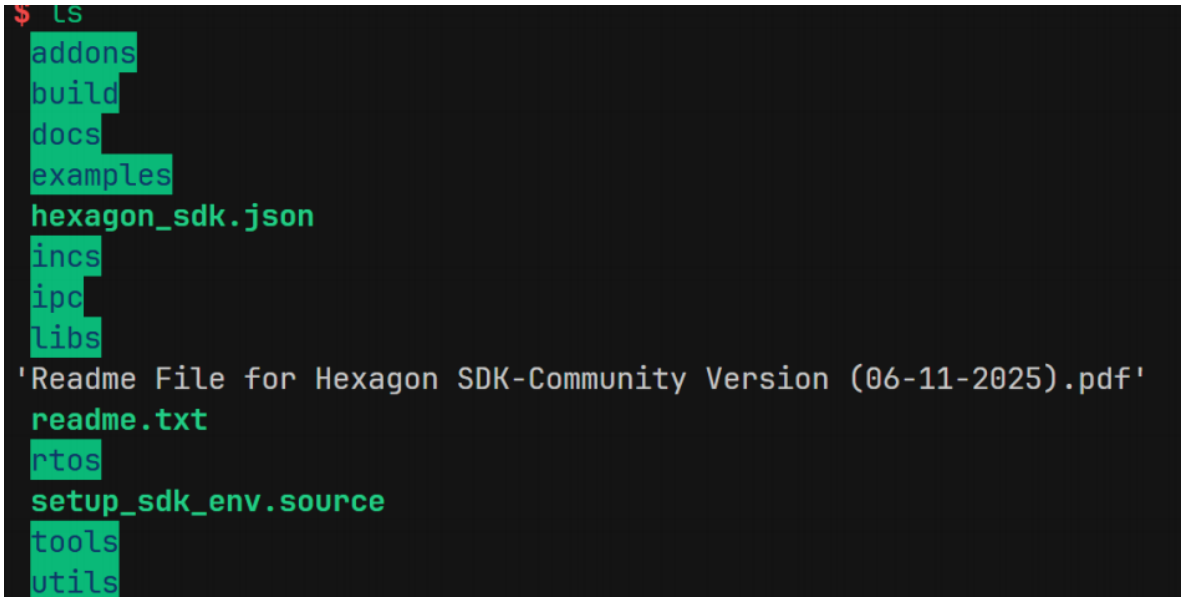
Lab4: Hexagon NPU GEMM

环境准备

Hexagon SDK 安装

```
wget https://apigwx-aws.qualcomm.com/qsc/public/v1/api/download/software/sdks/Hexagon_SDK/Linux/Debian/6.3.0.0/Hexagon_SDK.zip
unzip Hexagon_SDK.zip -d /workspace/Qualcomm/Hexagon_SDK/6.3.0.0
```



需要在 `~/.bashrc` 中设置 Hexagon SDK 目录的环境变量：

```
# 将下面的路径替换为实际的 Hexagon SDK 安装目录
export HEXAGON_SDK_PATH=/path/to/Qualcomm/Hexagon_SDK/6.3.0.0
```

保存后运行 `source ~/.bashrc` 或重开一个终端使配置生效。

- 在编译 Hexagon NPU 代码前需要执行 `source $HEXAGON_SDK_PATH/setup_sdk_env.source`

D. Codespaces / 权限 注意事项

如果你在 Codespaces 或受限环境中运行，可能需要调整 SDK 内部工具的权限：

```
sudo chown -R $(whoami):$(whoami) /path/to/Qualcomm/Hexagon_SDK/6.3.0.0/utils
```

将上面的 `$(whoami)` 替换为你的用户（例如 `codespace`），并将路径替换为实际安装目录。

Android NDK 安装

1. 下载并配置 Android SDK Command Line Tools

1. **下载 Command Line Tools**

```
wget https://googledownloads.cn/android/repository/commandlinetools-linux-13114758_latest.zip
```

2. **解压并组织目录结构**

```
unzip commandlinetools-linux-13114758_latest.zip
mkdir -p ~/Android/Sdk/cmdline-tools/latest
mv cmdline-tools/* ~/Android/Sdk/cmdline-tools/latest/
```

3. **配置环境变量**

   在 `~/.bashrc` 中添加：

```
export PATH=~/Android/Sdk/cmdline-tools/latest/bin/:$PATH
```

   使配置生效：

```
source ~/.bashrc
```

```
cmdline-tools  licenses  ndk
```

2. 安装 NDK

1. **查看可用版本并安装**

```
sdkmanager --list
sdkmanager "ndk;29.0.13113456"
```

```
system-images;android-34;google_atd;x86_64                          | 1
           | Google APIs ATD Intel x86_64 Atom System Image
system-images;android-35-ext14;google_apis_playstore;arm64-v8a      | 1
           | Google Play ARM 64 v8a System Image
system-images;android-35-ext14;google_apis_playstore;x86_64         | 1
           | Google Play Intel x86_64 Atom System Image
system-images;android-35-ext15;android-wear;arm64-v8a               | 1
           | Wear OS 5.1 ARM 64 v8a System Image
system-images;android-35-ext15;android-wear;x86_64                  | 1
           | Wear OS 5.1 Intel x86_64 Atom System Image
system-images;android-35-ext15;google_apis;arm64-v8a                | 1
           | Google APIs ARM 64 v8a System Image
system-images;android-35-ext15;google_apis;x86_64                   | 1
           | Google APIs Intel x86_64 Atom System Image
system-images;android-35-ext15;google_apis_playstore;arm64-v8a      | 1
           | Google Play ARM 64 v8a System Image
```

2. **配置 NDK 环境变量**

在 `~/.bashrc` 中添加：

```
export ANDROID_NDK_ROOT=~/Android/Sdk/ndk/29.0.13113456/
```

3. **安装 ADB 工具**

```
sudo apt install android-tools-adb
```

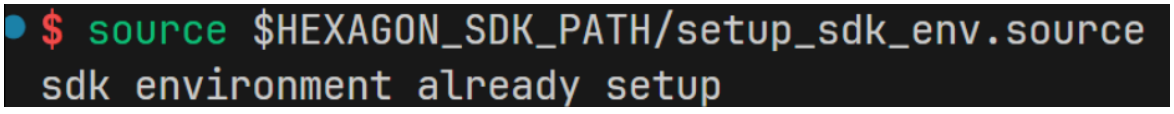
代码编译与运行

编译 NPU 代码

执行下列命令前，确保自己处于 `/OS-2026/Lab4` 目录

首先设置 Hexagon SDK 环境并编译 DSP 代码：

```
source $HEXAGON_SDK_PATH/setup_sdk_env.source
cd dsp
make hexagon BUILD=Debug DSP_ARCH=v79
```



实体设备运行 (推荐)

1. 申请 QDC 设备

1. 在 高通 QDC 平台 申请一台骁龙 8 Elite 手机
2. 选择 SSH 连接方式
3. 创建私钥并保存到 `~/qdc.pem`，修改权限：

```
chmod 600 ~/qdc.pem
```

2. 建立设备连接

1. **创建 SSH 隧道**

   点击 QDC 页面右上角的【Connect】按钮，复制连接命令，例如：

```
ssh -i ~/qdc.pem -L 5037:sa324277.sa.svc.cluster.local:5037 -N
sshtunnel@ssh.qdc.qualcomm.com
```

2. **验证设备连接**

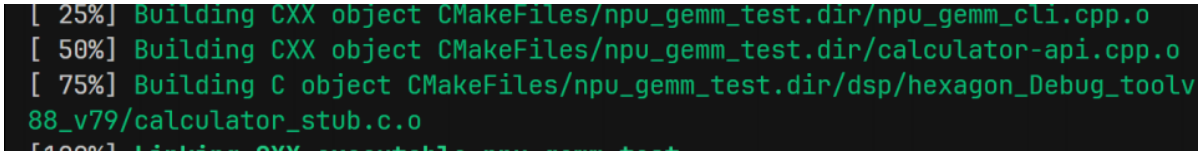
```
adb devices
```

   应该能看到已连接的设备。

3. 编译 Android 测试工具

执行下列命令前，确保自己处于 `/OS-2026/Lab4` 目录且 NPU 代码已经编译成功

```
source $HEXAGON_SDK_PATH/setup_sdk_env.source
mkdir build
cd build
cmake -DANDROID_ABI=arm64-v8a -DANDROID_PLATFORM=android-24 \
   -
DCMAKE_TOOLCHAIN_FILE=$ANDROID_NDK_ROOT/build/cmake/android.toolchain.cmake \
   -DHEXAGON_SDK_ROOT=$HEXAGON_SDK_ROOT ..
make
```



4. 部署并运行测试

执行下列命令前，确保自己处于 `/OS-2026/Lab4/build` 目录

1. **推送文件到设备**

```
adb push npu_gemm_test /data/local/tmp/
adb push ../dsp/hexagon_Debug_toolv88_v79/ship/libcalculator_skel.so
/data/local/tmp/
```

```
$ adb push npu_gemm_test /data/local/tmp/
adb push ../dsp/hexagon_Debug_toolv88_v79/ship/libcalculator_skel.so /data/lo
cal/tmp/
npu_gemm_test: 1 file pushed. 1.7 MB/s (2655408 bytes in 1.524s)
../dsp/hexagon_Debug_toolv88_v79/... 0.1 MB/s (31504 bytes in 0.251s)
```

2. **在设备上执行测试**

```
adb shell
cd /data/local/tmp
chmod +x npu_gemm_test
./npu_gemm_test 64 64 64 --cpu-check
```

实验编号	实现方式	设备/模拟器	矩阵尺寸 (M×K×N)	计算耗时 (ms)	备注
1	朴素 baseline		64×64×64	49.633	
2	HVX 内积 (A * B^T)		64×64×64	36.412	
3	HVX 外积 (A * B)		64×64×64	33.611	
4	朴素 baseline		256×256×256	468.286	
5	HVX 内积 (A * B^T)		256×256×256	90.973	
6	HVX 外积 (A * B)		256×256×256	61.649	
7	朴素 baseline		512×512×512	3593.954	
8	HVX 内积 (A * B^T)		512×512×512	378.946	
9	HVX 外积 (A * B)		512×512×512	290.618	
10	朴素 baseline		88×99×66	57.659	
11	HVX 内积 (A * B^T)		88×99×66	39.416	
12	HVX 外积 (A * B)		88×99×66	34.123	

```
Check if the result is correct...
CPU Result matrix C (first 10 elements):
0.13 0.14 0.21 0.12 0.15 0.19 0.12 0.18 0.16 0.12
✓ NPU and CPU results match within tolerance (by percent error)
sun:/data/local/tmp # ./npu_gemm_test 256 256 256 --cpu-check

================================
Starting NPU GEMM test with dimensions: C(m,n) = A(m,k) * B^T(n,k)
M=256, K=256, N=256
Initializing input matrices...
Calling NPU for GEMM calculation (A * B^T)...

[SUCCESS] NPU GEMM (A * B^T) calculation finished successfully!
NPU Result matrix C (first 10 elements):
0.97 0.77 0.89 0.76 0.84 0.81 0.81 0.89 0.79 0.94

Total time spent on NPU call: 92.823 ms

Check if the result is correct...
CPU Result matrix C (first 10 elements):
0.97 0.77 0.89 0.76 0.84 0.81 0.81 0.89 0.79 0.94
✓ NPU and CPU results match within tolerance (by percent error)
```

```
Starting NPU GEMM test with dimensions: C(m,n) = A(m,k) * B^T(n,k)
M=512, K=512, N=512
Initializing input matrices...
Calling NPU for GEMM calculation (A * B^T)...

[SUCCESS] NPU GEMM (A * B^T) calculation finished successfully!
NPU Result matrix C (first 10 elements):
1.77 1.80 1.83 1.86 1.77 1.80 1.83 1.86 1.77 1.80

Total time spent on NPU call: 290.618 ms
```

```
================================
Starting NPU GEMM test with dimensions: C(m,n) = A(m,k) * B^T(n,k)
M=64, K=64, N=64
Initializing input matrices...
Calling NPU for GEMM calculation (A * B^T)...

[SUCCESS] NPU GEMM (A * B^T) calculation finished successfully!
NPU Result matrix C (first 10 elements):
0.13 0.14 0.21 0.12 0.15 0.19 0.12 0.18 0.16 0.12

Total time spent on NPU call: 34.444 ms

Check if the result is correct...
CPU Result matrix C (first 10 elements):
0.13 0.14 0.21 0.12 0.15 0.19 0.12 0.18 0.16 0.12
✓ NPU and CPU results match within tolerance (by percent error)
/npu_gemm_test 1024 1024 1024 --cpu-check
```

特性	HVX 内积实现 (A * B^T)	HVX 外积实现 (A * B)
数据复用	- **A的一行**被重复读取 N 次，以计算该行与B的每一列的点积。 - **B'的一行**仅被读取一次，用于计算C的一个元素。	- **A的一个元素**被广播并重复使用 N/32 次。 - **C的一行**中的一个向量块被反复加载、更新、写回 K 次，数据局部性非常好。
内存访问模式	- 对A和B'的访问都是**连续的、流式的**，非常有利于硬件预取和缓存。	- 对A的访问是连续的。 - 对B的访问是**跨步的**，每次跳跃 N 个元素去取下一行对应的向量段，缓存效率相对较低。
向量指令使用	主要使用向量乘法 (vmpy) 和加法 (vadd)。其显著特点是最后需要一个**向量归约 (reduction)** 步骤，这涉及多次 vror 旋转和累加，会带来额外的指令开销。	主要使用**标量广播 (vsplat)** 和向量乘加。计算流水线更平滑，没有归约开销。

关键HVX 指令详解：

- **Q6_V_vzero()**: 初始化向量寄存器为零状态，作为归纳变量（累加器）的起始值，是所有累加操作的基础。
- **Q6_V_vsplat_R()**: 广播（Broadcast）指令。在外积法中，该指令将一个标量操作数复制到向量寄存器的所有通道，是实现标量-向量运算的关键。
- **Q6_Vqf32_vmpy_VsfVsf() / Q6_Vqf32_vadd_Vqf32Vqf32()**: 向量化算术指令。这些指令以 SIMD 方式执行计算，并将中间结果累加到高精度（QF32）格式的累加器中，以防止计算过程中的溢出和精度损失。
- **Q6_V_vror_VR()**: 向量旋转（Shuffle）指令。在内积法中，它被用于实现一种高效的并行归约算法。通过多次蝶式（butterfly-style）的旋转-相加操作，将向量内的部分和进行横向累加。
- **Q6_Vsf_equals_Vqf32()**: 格式转换指令。负责将高精度的 QF32 累加结果转换（或称为“去累积”）为标准的 IEEE 754 单精度浮点（SF）格式，以便写回主存。

针对尾部、对齐、缓存与内存带宽瓶颈提出优化建议。

- **对齐（Alignment）**：HVX 的向量加载/存储指令要求操作数地址为128字节对齐。本实现通过 memalign 保证了内存分配的对齐。对于非对齐的输入，必须通过 memcpy 拷贝至对齐的暂存区，这会引入额外的数据移动开销。
- **边界处理（Edge Case Handling）**：对于不能被向量长度（32）整除的维度，本实现退化为标量代码路径（scalar epilogue）进行处理。这保证了功能的正确性，但牺牲了性能。更优化的方法是采用**谓词执行（predicated execution）**，利用谓词寄存器（掩码）来禁用向量通道的写操作，从而在单次向量指令中处理非对齐的尾部数据，避免了分支和串行代码的开销。
- **缓存优化（Cache Optimization）**：对于超出 L1/L2 缓存容量的大型矩阵，性能瓶颈将从计算单元转移至内存子系统。解决此问题的标准技术是**缓存分块/循环平铺（Cache Blocking / Loop Tiling）**。该技术将计算任务分解为一系列能完全装入缓存的子问题（例如，对32x32的子矩阵进行乘法）。这极大地增强了时间局部性，减少了缓存未命中率和对主存总线的带宽需求，是实现极致性能的关键。

l2fetch(Rs,Rt)指令

Cache instructions (user-level)

Syntax	Permitted in packet	Operation
icinva(Rs)	Solo[4]	Instruction cache invalidate. Look up instruction cache at address Rs. If the address is in the cache, invalidate it.
dccleaninva(Rs)	slot 1 empty or ALU32 only	Data cache clean and invalidate. Look up data cache at address Rs. If the address is in the cache and has dirty data, flush that data out to memory. The cache line is then invalidated, whether or not dirty data was written.
dccleana(Rs)	slot 1 empty or ALU32 only	Data cache clean. Look up data cache at address Rs. If the address is in the cache and has dirty data, flush that data out to memory.
dcinva(Rs)	slot 1 empty or ALU32 only	Maps to dccleaninva(Rs).
dcfetch(Rs)	Normal[5]	Data cache prefetch. Prefetch data at address Rs into the data cache. **Note:** This instruction does not cause an exception.
l2fetch(Rs,Rt)	ALU32 or XTYPE only	L2 cache prefetch. Prefetch data from memory specified by Rs and Rt into L2 cache.