Run DeepSeek R1 Distill Model SG2042 RVV Comparison

测试小队

Feb 18, 2025

Section 1

测试简介

测试目的

本次测试旨在对比 SG2042 上不同指令集下的 DeepSeek R1 Distill Model 的性能。测试通过运行 `llama-bench` 得到结果。将会对比以下差异： - 线程数在不同指令集间的影响 - 不同指令集下的运行时间对比

下载模型

本次示例采用了基于 SG2042 的 MilkV-Pioneer 服务器进行，配置如下：

```
OS: Debian GNU/Linux trixie trixie/sid riscv64
RevyOS-Release: 20241025-001347
Host: Sophgo Mango
Kernel: Linux 6.6.56-pioneer
CPU: rv64gc (64)
Arch: rv64imafdc_zfh_zfhmin_zicbop_zicsr_zifencei_zihintpause_zmmul
  _xtheadba_xtheadbb_xtheadbs_xtheadcondmov_xtheadfmemidx_xtheadmac
  _xtheadmemidx_xtheadmempair_xtheadsync_xtheadvector
GPU: AMD Device 6779 (VGA compatible) [Discrete]
Memory: 120.98 GiB
Swap: Disabled
Disk (/): 365.00 GiB / 937.56 GiB (39%) - ext4
Locale: en_US.UTF-8
```

Section 2

搭建测试环境

获取模型

下载模型可以直接在网页端进行下载，也可以使用一个 Python 脚本进行下载。需要使用 pip 安装：

```
pip install huggingface_hub hf_transfer
```

```
import os

from huggingface_hub import snapshot_download
snapshot_download(
  repo_id = "unsloth/DeepSeek-R1-Distill-Llama-8B-GGUF",
  local_dir = "DeepSeek-R1-Distill-Llama-8B-GGUF",
  allow_patterns = ["*Q4_K_M*", "*Q2_K*"],
)
```

自行更改 repo_id 和 local_dir 即可。

我们使用了 DeepSeek-R1-Distill-Llama-8B-GGUF Q4_K_M 和 DeepSeek-R1-Distill-Qwen-1.5B-GGUF 这两个模型的 Q4_K_M 和 Q2_K 量化模型。

自动脚本进行

为了保证测试的一致性和简易性，我们为此次测试编写了脚本，可以自动初始化环境、编译并

脚本可见于仓库 deepseek_sg2042_compare 下的 run.sh。

简单用法如下：

- ./run.sh -h：查看帮助
- 编译并运行测试 ./run.sh -a -b -m [model] -t [threads]
- 不编译直接运行测试 ./run.sh -a -m [model] -t [threads]

更多用法请查看帮助。

手动搭建测试环境

由于不同指令集间的环境搭建大部分相同，下面将会以 rv64gc 为例进行说明，并标注对于 rv64gcv0p7 的不同之处。

Toolchain

获取专用的toolchain。对应的toolchain如下：

- rv64gc：RuyiSDK-20231212-Upstream-Sources-HOST-riscv64-linux-gnu-riscv64-unknown-linux-gnu
- rv64gcv0p7：RuyiSDK-20240222-T-Head-Sources-T-Head-2.8.0-HOST-riscv64-linux-gnu-riscv64-plctxthead-linux-gnu

下载后解压，将工具链路径加入环境变量：

```
wget "https://mirror.iscas.ac.cn/ruyisdk/dist/RuyiSDK-20231212-Upstream-Sources
tar -xvf "RuyiSDK-20231212-Upstream-Sources-HOST-riscv64-linux-gnu-riscv64-unkn
mv "RuyiSDK-20231212-Upstream-Sources-HOST-riscv64-linux-gnu-riscv64-unknown-li
expor PATH=/path/to/upstream_toolchain/bin:$PATH
```

对于 rv64gcv0p7，工具链为 RuyiSDK-20240222-T-Head-Sources-T-Head-2.8.0-HOST-riscv64-linux-gnu-riscv64-plctxthead-linux-gnu，自行下载并解压即可。

后续再次使用需要将工具链重新加入 PATH 中。

注意事项：

- 以上工具链使用的 sysroot
  并非系统的根目录，而是工具链自带的。在安装库时需要注意路径。
- 在使用时需要手动设置 CC=riscv64-plctxthead-linux-gnu-gcc 和 riscv64-plctxthead-linux-gnu-gfortran

OpenBLAS 后端

目前 Llama.cpp 支持的后端中，只有 OpenBLAS 有 RVV0p7 的支持。因此，采用 OpenBLAS 作为 GGML 的后端。

```
git clone https://github.com/OpenMathLib/OpenBLAS
cd OpenBLAS
make HOSTCC=riscv64-unknown-linux-gnu-gcc TARGET=RISCV64_GENERIC \
  CC=riscv64-unknown-linux-gnu-gcc FC=riscv64-unknown-linux-gnu-gfortran
sudo make install PREFIX=/usr/local
sudo make install PREFIX=~/
```

注意需要在两个地方安装，一个是系统级的 /usr/local，一个是工具链的 sysroot。若使用 rv64gcv0p7，则需要在编译时指定 TARGET=C910V，同时更改 CC FC 前缀为 riscv64-plctxthead-linux-gnu-。

Llama.cpp

直接从 GitHub 上获取最新的 llama.cpp 源码：

```
git clone https://github.com/ggerganov/llama.cpp.git
cd llama.cpp
```

首先需要 patch llama.cpp，让其使用正确的架构。patch 文件如下：

```
diff --git a/ggml/src/ggml-cpu/CMakeLists.txt b/ggml/src/ggml-cpu/CMakeLists.tx
index 98fd18e..0e6f302 100644
--- a/ggml/src/ggml-cpu/CMakeLists.txt
+++ b/ggml/src/ggml-cpu/CMakeLists.txt
@@ -306,7 +306,7 @@ function(ggml_add_cpu_backend_variant_impl tag_name)
     elseif (${CMAKE_SYSTEM_PROCESSOR} MATCHES "riscv64")
         message(STATUS "RISC-V detected")
         if (GGML_RVV)
-            list(APPEND ARCH_FLAGS -march=rv64gcv -mabi=lp64d)
+            list(APPEND ARCH_FLAGS -march=rv64gc -mabi=lp64d)
         endif()
     else()
         message(STATUS "Unknown architecture")
```

对于 RVV0p7，需要将 rv64gcv 改为 rv64gcv0p7。

直接 apply patch：patch -p1 < patch.diff

指定 OpenBLAS 进行编译，注意设置 CC 和 FC：

```
CC=riscv64-unknown-linux-gnu-gcc FC=riscv64-unknown-linux-gnu-gfortran \
cmake -B build -DGGML_BLAS=ON -DGGML_BLAS_VENDOR=OpenBLAS
cmake --build build --config Release -j32
```

使用 rv64gcv0p7 时，需要将 CC FC 前缀改为 riscv64-plctxthead-linux-gnu-。

运行测试

直接使用 llama.cpp 提供的 llama-bench
进行测试。为了避免使用错误的后端，在此建议将 libopenblas.so
移动到程序目录，并删除系统中的 libopenblas.so。 在此移动位于工具链的 sysroot
下的 libopenblas.so:

```
find /path/to/upstream_toolchain/riscv64-unknown-linux-gnu/sysroot \
  -name "libopenblas*" | xargs -I {} cp {} llama.cpp/build/bin
find /usr/local/lib -name "libopenblas*" | xargs sudo rm
```

进入 llama.cpp/build/bin 目录，运行测试：

```
cd llama.cpp/build/bin
./llama-bench \
  -m ../../DeepSeek-R1-Distill-Qwen-1.5B-GGUF/DeepSeek-R1-Distill-Qwen-1.5B-Q4_
  -t 32 --progress
```

更改模型和线程数即可。

Section 3

测试结果

同一指令集间对比

线程数/test