Application Note

为 TIDL 使用分配内存

Adam Hua, Sotirios 'Chris' Tsongas

摘要

TIDL 是 TI 的 AI 推理框架，在 TDA4x 系列和 AM6xA 系列处理器上运行，可利用内置的 AI 加速器 C7xMMA 实现高效的 AI 模型推理。TI 处理器采用共享的存储器架构，其中所有内核，包括运行高级操作系统 (Linux、QNX) 的 A 内核和用于 AI 推理的 C7xMMA，均共享同一个物理内存。其他内核无法访问预分配给 C7xMMA 的内存空间，当内存空间有限时，这可能会导致主系统的运行时内存不足。本文介绍了如何根据实际模型运行时条件预分配 C7xMMA 内存，以最大限度地减小其内存占用。

内容

商标

1 简介

TI 的 SoC 采用共享内存方法，这意味着 NPU 会与其他内核共享物理内存。在默认系统中，会根据 SoC 的资源为 NPU 保留相对较大的内存。保留的内存不能被其他内核使用，因此在内存受限的情况下，可能会因无法获取所需内存而导致其他进程执行失败，而 NPU 可能不会使用所有预分配内存空间，进而导致浪费宝贵资源。因此，我们需要根据模型的实际运行时条件为 C7xMMA 分配内存空间，以最大限度地提高资源利用率。

2 C7xMMA DDR 使用情况分析和优化

在电路板上运行模型时，设置调试级别可以在模型执行期间输出内存要求。使用 /opt/tidl_test/TI_DEVICE_armv8_test_dl_algo_host_rt.out 并将 debugTraceLevel 设置为 2，或使用 edgeai-tidl-tools 并将 debug_level 设置为 2，或在使用 TIOVX 应用时，在 TIDL_CreateParams 中将 traceLogLevel 设置为 2。在进行模型推理之前，运行 /opt/vision_apps/vision_apps_init.sh 以启用 C7xMMA 日志记录。此时，进行模型推理将生成日志，如下图所示。这些日志会说明为 TIDL 的各个部分分配的存储空间。

```
[C7x_2 ]     52.423052 s: 0           , DDR Non-cacheable   , Persistent  ,  128, 19.42   , 0x10000000
[C7x_2 ]     52.423071 s:
[C7x_2 ]     52.423106 s: 1           , DDR Cacheable       , Persistent  ,  128, 0.66    , 0x1043de40
[C7x_2 ]     52.423123 s:
[C7x_2 ]     52.423158 s: 2           , L1D                 , Scratch     ,  128, 16.00   , 0x7f83c000
[C7x_2 ]     52.423176 s:
[C7x_2 ]     52.423210 s: 3           , L2                  , Scratch     ,  128, 224.00  , 0x7f800000
[C7x_2 ]     52.423228 s:
[C7x_2 ]     52.423263 s: 4           , L3/MSMC             , Scratch     ,  128, 2048.00 , 0x7e200000
[C7x_2 ]     52.423281 s:
[C7x_2 ]     52.423316 s: 5           , DDR Cacheable       , Persistent  ,  128, 2755.41 , 0x1043e180
[C7x_2 ]     52.423333 s:
[C7x_2 ]     52.423368 s: 6           , DDR Non-cacheable   , Scratch     ,  128, 4.00    , 0x10300000
[C7x_2 ]     52.423386 s:
 Freeing memory for user provided Net
[C7x_2 ]     52.423421 s: 7           , DDR Non-cacheable   , Persistent  ,  128, 148.25  , 0x10005000
[C7x_2 ]     52.423439 s:
[C7x_2 ]     52.423473 s: 8           , DDR Non-cacheable   , Scratch     ,  128, 0.13    , 0x10301000
[C7x_2 ]     52.423492 s:
[C7x_2 ]     52.423526 s: 9           , DDR Non-cacheable   , Scratch     ,  128, 3.13    , 0x10301400
[C7x_2 ]     52.423544 s:
[C7x_2 ]     52.423578 s: 10          , DDR Cacheable       , Persistent  ,  128, 530.19  , 0x106eefc0
[C7x_2 ]     52.423596 s:
[C7x_2 ]     52.423631 s: 11          , DDR Cacheable       , Scratch     ,  128, 4096.25 , 0x10e00000
[C7x_2 ]     52.423649 s:
[C7x_2 ]     52.423684 s: 12          , DDR Non-cacheable   , Persistent  ,  128, 2048.00 , 0x1002a400
[C7x_2 ]     52.423701 s:
[C7x_2 ]     52.423736 s: 13          , DDR Cacheable       , Persistent  ,  128, 1390.06 , 0x10773900
[C7x_2 ]     52.423754 s:
[C7x_2 ]     52.423788 s: 14          , DDR Non-cacheable   , Persistent  ,  128, 0.00    , 0x1022a400
[C7x_2 ]     52.423806 s:
[C7x_2 ]     52.423840 s: 15          , DDR Cacheable       , Persistent  ,  128, 4381.13 , 0x108cf1c0
[C7x_2 ]     52.423857 s:
```

图 2-1. 模型运行时内存空间分配

该表会显示内存分配记录，即打印 memRec 信息。MemRec 中有 16 条内存记录，记录了模型运行期间不同进程占用的存储空间。下面的内容简要介绍了各个说明的用途：

录音	说明	可缓存	属性
0	TIDL 实例	否	永久性
1	TIDL_CreateParams 占用的空间	是	永久性
2	L1D SRAM	-	暂存
3	L2 SRAM	-	暂存
4	L3 SRAM	-	暂存
5	存储每层的固定输入，如权重	是	永久性
6	中间层输出结果（用于模型转换）	否	暂存
7	中间层输出结果（模型推理）	否	永久性
8	每个层临时使用的空间，用于累积输出值并取最大值。	否	暂存
9	存储统计信息和一些记录信息	否	暂存
10	每层的 sTIDL_AlgLayer_t 之和，记录每层的基本信息	是	永久性
11	每个层的算法应用使用的空间	是	暂存
12	在处理抢占时存储上下文	否	永久性

录音	说明	可缓存	属性
13	网络结构	是	永久性
14	用于多核模式下的多核同步	否	永久性
15	存储一些常量，例如卷积参数	是	永久性

为了提高 CPU 运行效率，应将 DDR 设置为尽可能可缓存。对于需要被其他内核访问的内存，必须将其设置为不可缓存，以避免多核系统中出现缓存一致性问题。持久化表示内存块中的数据长时间保持不变，而暂存则表示数据会频繁变化。

参考文档来配内存映射时，需要注意每次使用 C7 堆时都需要配置以下四种大小：

- C7x_x_ddr_local_heap_size，对应可缓存持久化内存块
- C7x_x_ddr_scratch_size，对应可缓存暂存内存块
- C7x_x_ddr_local_heap_non_cacheable_size，对应不可缓存持久化内存块
- C7x_x_ddr_scratch_non_cacheable_size，对应不可缓存暂存内存块

我们只需根据内存块是否可缓存，是持久化块还是暂存块，来对图 1 中的内存块进行分类和求和，即可获得需要为上述四个块配置的相应大小。在具有多个 C7xMMA 的芯片中，需要根据这种方法为每个 C7x 配置其自身的四个内存块。

读者可能还会注意到，在 gen_linker_mem_map.py 中，除了 C7x 堆外，还有一些名为 c7x_x_ddr_size 的内存块。这些内存块负责存储 IPC 通信、MCU 启动固件、基本运行时所需堆栈等，并可根据实际使用情况酌情减少。

请注意，配置内存映射时，请严格遵循文档中的内存对齐和最小内存要求。

3 测试基于 J722S 和 MobileNet

本文使用 SDK 版本 11.01 在 J722S EVM 上进行了测试。虽然其他 SDK 版本可能适用，但我们建议使用 SDK 11.01 并在 J722S EVM 上运行。

3.1 固件编译和环境配置

根据 SDK 用户指南配置环境并执行第一次编译。根据 TIDL 编译指南配置 TIDL 所需的环境，并执行第一次编译。

3.2 模型导出和板载推理

完成 3.1 后，将在 SDK 中安装默认测试模型 MobileNet。在 ti-processor-sdk-rtos-j722s-evm-11_01_00_04/c7x-mma-tidl/ti_dl/utils/tidlModelImport 中运行：

./out/tidl_model_import.out ../../test/testvecs/config/import/public/caffe/tidl_import_mobilenet_v1.txt

导入完成后，必须将模型工件复制到目标板。将导出的模型复制到电路板，在 infer.txt 中配置推理参数。infer.txt 文件应包含以下配置：

```
inFileFormat = 1
numFrames = 1
postProcType = 1
postProcDataId = 0
inData = input.bin
outData = "output.bin"
netBinFile = tidl_net_mobilenet_v1.bin
ioConfigFile = tidl_io_mobilenet_v1_1.bin
writeTraceLevel = 0
debugTraceLevel = 2
coreNum = 2
```

请注意，要设置 debugTraceLevel = 2。此 infer.txt 文件中的路径假设模型工件（tidl_net_mobilenet_v1.bin 和 tidl_io_mobilenet_v1_1.bin）、input.bin 与 TI_DEVICE_armv8_test_dl_algo_host_rt.out 可执行文件位于同一目录中。

然后运行：

/opt/tidl_test/TI_DEVICE_armv8_test_dl_algo_host_rt.out s:infer.txt

运行 /opt/vision_apps/vision_apps_init.sh 以获得图 1 中的结果。

3.3 内存统计信息

根据图 1 中的值执行内存统计，并获得下表：

内存类型	尺寸
可缓存持久化	9058.16KB
可缓存暂存	4096.25KB
不可缓存持久化	2215.67KB
不可缓存暂存	7.26KB

3.4 修改内存映射

按以下方式修改 gen_linker_mem_map.py：

```
diff --git a/platform/j722s/rtos/gen_linker_mem_map.py b/platform/j722s/rtos/gen_linker_mem_map.py
index 18a008d..2cf290b 100755
--- a/platform/j722s/rtos/gen_linker_mem_map.py
+++ b/platform/j722s/rtos/gen_linker_mem_map.py
@@ -236,18 +236,18 @@ c7x_1_ddr_scratch_size         = 64*MB;
 # C7x_2 Non-Cache sections
 c7x_2_ddr_local_heap_non_cacheable_addr       = c7x_1_ddr_scratch_addr + c7x_1_ddr_scratch_size;
 c7x_2_ddr_local_heap_non_cacheable_addr_phys  = c7x_1_ddr_scratch_addr_phys + c7x_1_ddr_scratch_size;
-c7x_2_ddr_local_heap_non_cacheable_size       = 64*MB;
+c7x_2_ddr_local_heap_non_cacheable_size       = 3*MB;
 c7x_2_ddr_scratch_non_cacheable_addr          = c7x_2_ddr_local_heap_non_cacheable_addr + c7x_2_ddr_local_heap_non_cacheable_size;
 c7x_2_ddr_scratch_non_cacheable_addr_phys     = c7x_2_ddr_local_heap_non_cacheable_addr_phys + c7x_2_ddr_local_heap_non_cacheable_size;
-c7x_2_ddr_scratch_non_cacheable_size          = 64*MB;
+c7x_2_ddr_scratch_non_cacheable_size          = 1*MB;

 # C7x_2 Cache sections
 c7x_2_ddr_local_heap_addr      = c7x_2_ddr_scratch_non_cacheable_addr + c7x_2_ddr_scratch_non_cacheable_size;
 c7x_2_ddr_local_heap_addr_phys = c7x_2_ddr_scratch_non_cacheable_addr_phys + c7x_2_ddr_scratch_non_cacheable_size;
-c7x_2_ddr_local_heap_size      = 64*MB;
+c7x_2_ddr_local_heap_size      = 10*MB;
 c7x_2_ddr_scratch_addr         = c7x_2_ddr_local_heap_addr + c7x_2_ddr_local_heap_size;
 c7x_2_ddr_scratch_addr_phys    = c7x_2_ddr_local_heap_addr_phys + c7x_2_ddr_local_heap_size;
-c7x_2_ddr_scratch_size         = 64*MB;
+c7x_2_ddr_scratch_size         = 5*MB;

 total_c7x_1_local_ddr   = c7x_1_ddr_local_heap_non_cacheable_size + c7x_1_ddr_local_heap_size;
 total_c7x_1_scratch_ddr = c7x_1_ddr_scratch_non_cacheable_size + c7x_1_ddr_scratch_size
```

图 3-1. 内存映射修改

此修改基于第 3.3 节中的统计结果，考虑该模型仅在第二个 C7xMMA 上运行的情况。因此，仅修改与 C7x_2 对应的堆区段内存。其中，可缓存持久化占用 9058.16KB。在配置中，SDK 使用 1000（而不是 1024）进行 KB 到 MB 的转换。此外，考虑到 1MB 内存对齐要求，将 c7x_2_ddr_local_heap_size 配置为 10MB。可缓存暂存占用 4096.25KB，考虑到 1MB 对齐要求，将 c7x_2_ddr_scratch_size 配置为 5MB。不可缓存持久化占用 2215.67KB，c7x_2_ddr_local_heap_non_cacheable_size 配置为 3MB。不可缓存暂存占用 7.26KB，c7x_2_ddr_scratch_non_cacheable_size 配置为 1MB。

Sysconfig 的修改方式如下：

```
diff --git a/platform/j722s/rtos/c7x_2/example.syscfg b/platform/j722s/rtos/c7x_2/example.syscfg
index de7cb27..23cdd1f 100644
--- a/platform/j722s/rtos/c7x_2/example.syscfg
+++ b/platform/j722s/rtos/c7x_2/example.syscfg
@@ -234,7 +234,7 @@ mmu_armv814.attribute          = "MAIR7";
 mmu_armv815.$name              = "DDR_C7X_2_HEAP_SCRATCH_NON_CACHEABLE";
 mmu_armv815.vAddr              = 0x110000000;
 mmu_armv815.pAddr              = 0x890000000;
-mmu_armv815.size               = 0x8000000;
+mmu_armv815.size               = 0x400000;
 mmu_armv815.attribute          = "MAIR4";

 /**
@@ -245,7 +245,7 @@ mmu_armv815.attribute          = "MAIR4";
  * Attribute is MAIR7.
  */
 mmu_armv816.$name              = "DDR_C7X_2_HEAP_SCRATCH_CACHEABLE";
-mmu_armv816.vAddr              = 0x118000000;
-mmu_armv816.pAddr              = 0x898000000;
-mmu_armv816.size               = 0x8000000;
+mmu_armv816.vAddr              = 0x110400000;
+mmu_armv816.pAddr              = 0x890400000;
+mmu_armv816.size               = 0xF00000;
 mmu_armv816.attribute          = "MAIR7";
```

图 3-2. Sysconfig 内存配置修改

有关实现内存映射修改的说明，请参阅文档。

3.5 重新编译 SDK 并更新到电路板

根据 SDK 用户指南执行 RTOS SDK 编译和更新。根据 Linux SDK 用户指南编译 U-Boot 并更新 tiboot3.bin。

3.6 板载测试

将 tidl_net_mobilenet_v1.bin、tidl_io_mobilenet_v1_1.bin、infer.txt 和 input.bin 文件复制到设备上的 /opt/tidl_test。

在电路板上运行：

```
cd /opt/tidl_test/
./TI_DEVICE_armv8_test_dl_algo_host_rt.out s:infer.txt
```

获取测试结果：

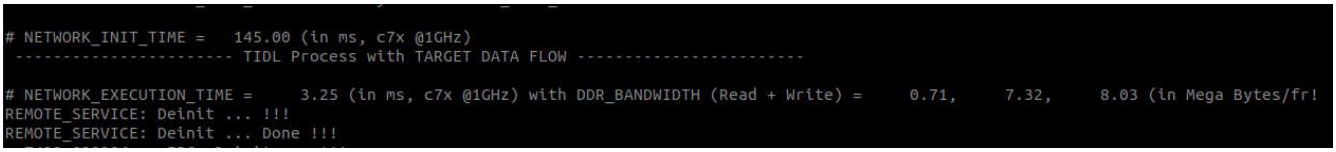

图 3-3. 板载推理结果

4 总结

本文讨论了 AI 模型在推理期间的实际内存使用情况，描述了每个内存块的用途，并提供了有关如何根据实际使用情况合理分配内存空间以节省 DDR 空间资源的指导。本文还提供了一个测试用例，使用 SDK 随附的模型进行测试，并成功将单个 C7x 占用的内存从 256MB 减少到 19MB，从而有效地减少了内存使用并降低了整个电路板的成本。

5 参考资料

1. https://www.ti.com.cn/product/cn/AM62A7
2. https://www.ti.com/tool/PROCESSOR-SDK-AM62A
3. https://www.ti.com/product/AM67A
4. https://www.ti.com.cn/product/cn/TDA4VM
5. https://www.ti.com/product/TDA4VE-Q1
6. https://www.ti.com/product/TDA4VM-Q1
7. https://www.ti.com/product/TDA4AL-Q1
8. https://www.ti.com/product/TDA4VL-Q1
9. https://www.ti.com/product/TDA4VP-Q1
10. https://www.ti.com/product/TDA4VH-Q1
11. https://www.ti.com/product/TDA4VEN-Q1

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月