参考資料

Application Note

TIDL 使用時のメモリの割り当て

Adam Hua, Sotirios 'Chris' Tsongas

概要

TIDL は、TDA4x シリーズと AM6xA シリーズの各プロセッサ上で動作する TI の AI 推論フレームワークであり、内蔵 AI アクセラレータ C7xMMA を活用して効率的な AI モデル推論を実現します。TI のプロセッサは共有メモリアーキテクチャを採用しており、ハイレベルオペレーティングシステム (Linux、QNX) を実行する A コアや AI 推論用の C7xMMA を含むすべてのコアが同じ物理メモリを共有します。C7xMMA に事前割り当てされたメモリ空間は、他のコアからアクセスできないため、メモリ空間が制限されている場合、メインシステム用のランタイムメモリが不十分になる可能性があります。本書では、C7xMMA メモリを実際のモデル実行時の条件に基づいて事前に割り当ててメモリ占有量を最小化する方法について述べています。

目次

1 はじめに....2
2 C7xMMA DDR 使用状況の分析と最適化....3
3 J722S と MobileNet に基づくテスト....5
3.1 ファームウェアのコンパイルと環境設定....5
3.2 モデルのエクスポートとオンボード推論....5
3.3 メモリ統計....5
3.4 メモリマップの変更....5
3.5 SDK の再コンパイルとボードへの更新....7
3.6 オンボードテスト....7
4 まとめ....8
5 参考資料....9

商標

English Document: SDAA177

1 はじめに

TI の SoC は共有メモリアプローチを採用しています。つまり、NPU は他のコアと物理メモリを共有します。デフォルトのシステムでは、SoC のリソースに基づいて NPU 用に比較的大量のメモリが予約されています。予約済みメモリは他のコアでは使用できないため、メモリに制約がある状況では、必要なメモリを取得できないために他のプロセスが失敗する可能性があります。一方、NPU は事前に割り当てられたメモリ空間を完全に使用できず、貴重なリソースを浪費する可能性があります。したがって、リソース利用率を最大化するためには、モデルの実際の実行時条件に応じて C7xMMA にメモリ空間を割り当てる必要があります。

2 C7xMMA DDR 使用状況の分析と最適化

ボード上でモデルを実行する際にデバッグレベルを設定すると、モデルの実行中にメモリ要件を出力できます。/opt/tidl_test/TI_DEVICE_armv8_test_dl_algo_host_rt.out を使用して debugTraceLevel を 2 に設定するか、edgeai-tidl-tools を使用して debug_level を 2 に設定する、または TIOVX アプリケーションを使用する際に TIDL_CreateParams で traceLogLevel を 2 に設定します。モデルを推測する前に、/opt/vision_apps/vision_apps_init.sh を実行して C7xMMA ロギングを有効にします。この時点で、モデルを推測すると、下図のようなログが生成されます。これらのログは、TIDL の各部分に割り当てられるストレージ容量を示しています。

```
[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:
```

図 2-1. モデル実行時におけるメモリ空間の割り当て

この表は、メモリ割り当てレコード、つまり memRec 情報の印刷を示しています。MemRec には 16 個のメモリレコードがあり、モデル操作中にさまざまなプロセスが占有する記憶領域を記録します。以下では、各説明の目的について簡単に説明します。

記録	説明	キャッシュ可否	属性
0	TIDL インスタンス	なし	永続
1	TIDL_CreateParams により占有されている空間	あり	永続
2	L1D SRAM	-	スクラッチ
3	L2 SRAM	-	スクラッチ
4	L3 SRAM	-	スクラッチ
5	重みなど、各レイヤの固定入力を格納	あり	永続
6	中間層の出力結果 (モデル変換用)	なし	スクラッチ
7	中間層の出力結果 (モデル推論)	なし	永続
8	各レイヤに一時的に使用される領域で、出力値を蓄積し、最大値を取得するために使用。	なし	スクラッチ
9	統計情報と一部の記録情報を保存	なし	スクラッチ
10	各レイヤの sTIDL_AlgLayer_t の合計で、各レイヤの基本情報を記録	あり	永続
11	各レイヤのアルゴリズム処理で使用される領域	あり	スクラッチ

記録	説明	キャッシュ可否	属性
12	プリエンプションを処理する際にコンテキストを格納	なし	永続
13	ネットワーク構造	あり	永続
14	マルチコアモードでのマルチコア同期に使用	なし	永続
15	畳み込みパラメータなどの定数を格納	あり	永続

CPU の動作効率を向上させるためには、DDR を可能な限りキャッシュ可能に設定する必要があります。他のコアからアクセスする必要があるメモリについては、マルチコアシステムでキャッシュコヒーレンシの問題を回避するために、キャッシュ不可で設定する必要があります。永続は、メモリブロック内のデータが長期間変更されないことを示し、スクラッチはデータが頻繁に変化することを示します。

マニュアルを参照してメモリマップを構成する場合、C7 ヒープを使用するごとに次の 4 つのサイズを設定する必要があることに注意してください。

- c7x_x_ddr_local_heap_size は、キャッシュ可能の永続メモリブロックに対応します
- c7x_x_ddr_scratch_size は、キャッシュ可能のスクラッチメモリブロックに対応します
- c7x_x_ddr_local_heap_non_cacheable_size は、キャッシュ不可の永続メモリブロックに対応します
- c7x_x_ddr_scratch_non_cacheable_size は、キャッシュ不可のスクラッチメモリブロックに対応します

図 1 のメモリブロックを、キャッシュ可能で永続的かスクラッチかどうかに応じて分類して合計するだけで、上記の 4 つのブロックに対して設定する必要があるそれぞれのサイズを取得できます。複数の C7xMMA を搭載したチップでは、各 C7x はこの方法に従って独自の 4 つのメモリブロックを設定する必要があります。

また、gen_linker_mem_map.py 内に C7x ヒープに加えて、c7x_x_x_ddr_size という名前のメモリブロックがいくつか存在することに気付くかもしれません。これらのメモリブロックは、IPC 通信、MCU 起動用ファームウェア、基本的なランタイムが必要なスタックなどを保存します。実際の使用方法に基づいて、必要に応じて数を減らすことができます。

メモリマップを設定する場合は、ドキュメントに記載されているメモリのアライメントと最小メモリの要件に厳密に従ってください。

3 J722S と MobileNet に基づくテスト

この記事は、SDK バージョン 11.01 を使用して、J722S EVM でテスト済みです。他の SDK バージョンも動作する可能性がありますが、SDK 11.01 を使用し、J722S 評価基板上で実行することをお勧めします。

3.1 ファームウェアのコンパイルと環境設定

SDK ユーザーガイドに従って、環境を構成し、最初のコンパイルを実行します。TIDL が必要とする環境を構成し、TIDL Compilation Guide に従って最初のコンパイルを実行します。

3.2 モデルのエクスポートとオンボード推論

3.1 を完了した後、デフォルトのテストモデルである MobileNet を SDK にインストールします。ti-processor-sdk-rtos-j722s-evm-11_01_00_04/c7x-mma-tidl/ti_dl/utils/tidlModelImport を実行:

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

インポートが完了したら、モデルアーティファクトをターゲットボードにコピーする必要があります。エクスポートしたモデルをボードにコピーし、infer.txt で推論パラメータを構成します。infer.txt ファイルには、以下の構成内容を記載する必要があります。

```
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
```

debugTraceLevel = 2 を設定することに注意してください。この infer.txt ファイルのパスは、モデルアーティファクト (tidl_net_mobilenet_v1.bin および tidl_io_mobileenet_v1_1.bin) を想定し、input.bin は TI_DEVICE_armv8_test_dl_algo_host_rt.out 実行ファイルと同じディレクトリにあります。

次のように実行します。

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

/opt/vision_apps/vision_apps_init.sh を実行して、図 1 の結果を取得します。

3.3 メモリ統計

図 1 の値に従ってメモリ統計情報を実行し、次の表を取得します。

メモリ タイプ	サイズ
キャッシュ可の永続領域	9058.16KB
キャッシュ可のスクラッチ領域	4096.25KB
キャッシュ不可の永続	2215.67KB
キャッシュ不可のスクラッチ領域	7.26KB

3.4 メモリマップの変更

gen_linker_mem_map.py を次のように変更します。

```
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
```

図 3-1. メモリマップの変更

この変更は、モデルが 2 番目の C7xMMA 上でのみ実行されることを考慮して、セクション 3.3 の統計結果に基づいています。したがって、C7x_2 に対応するヒープセグメントメモリのみが変更されます。その中で、キャッシュ可の永続領域は 9058.16 KB を占めています。設定内では、SDK は KB から MB への換算に 1024 ではなく 1000 を使用します。さらに、1MB メモリアライメント要件を考慮して、c7x_2_ddr_local_heap_size が 10MB に設定されています。キャッシュ可のスクラッチ領域は 4096.25KB を占有し、1MB アライメント要件を考慮して、c7x_2_ddr_scratch_size を 5MB に設定します。キャッシュ不可の永続領域は 2215.67 KB を占有し、c7x_2_ddr_local_heap_non_cacheable_size は 3MB に設定されます。キャッシュ不可のスクラッチ領域は 7.26 KB を占有し、c7x_2_ddr_scratch_non_cacheable_size は 1MB に設定されています。

Sysconfig の変更は次のとおりです。

```
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";
```

図 3-2. Sysconfig メモリ設定の変更

メモリマップの変更を実施する手順については、ドキュメントを参照してください。

3.5 SDK の再コンパイルとボードへの更新

SDK ユーザーガイドに従って、RTOS SDK のコンパイルと更新を実行します。Linux SDK ユーザーガイドに従って、U-Boot をコンパイルし、tiboot3.bin を更新してください。

3.6 オンボードテスト

tidl_net_mobileenet_v1.bin、tidl_io_mobileenet_v1_1.bin、infer.txt、および input.bin ファイルをデバイスの /opt/tidl_test にコピーします。

オンボードで動作:

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

テスト結果を取得:

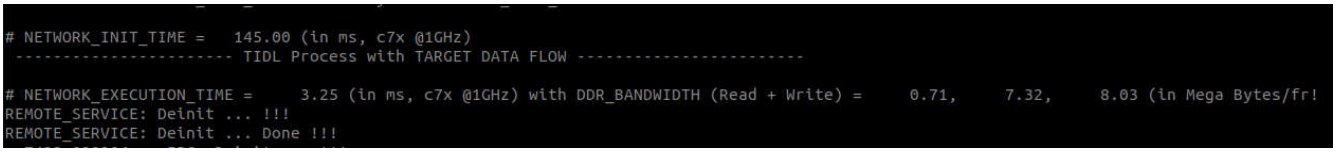

図 3-3. オンボードの推論結果

4 まとめ

本稿では、推論中の AI モデルの実際のメモリ使用量を解説し、各メモリブロックの目的を説明するとともに、DDR 空間リソースを節約するために実際の使用状況に基づいてメモリ空間を合理的に割り当てる方法についてのガイダンスを提供します。この記事では、SDK に付属するモデルを使用したテストケースも紹介しており、単一の C7x で使用するメモリを 256MB から 19MB に正常に削減することができます。その結果、メモリ使用量を効果的に低減し、ボード全体のコストを削減することができます。

5 参考資料

1. https://www.ti.com/product/AM62A7
2. https://www.ti.com/tool/PROCESSOR-SDK-AM62A
3. https://www.ti.com/product/AM67A
4. https://www.ti.com/product/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

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月