NEST-C의 파티션 튜너 사용 방법

2022. 12. 20.

유미선

한국전자통신연구원 인공지능프로세서/SW연구실

문서 이력

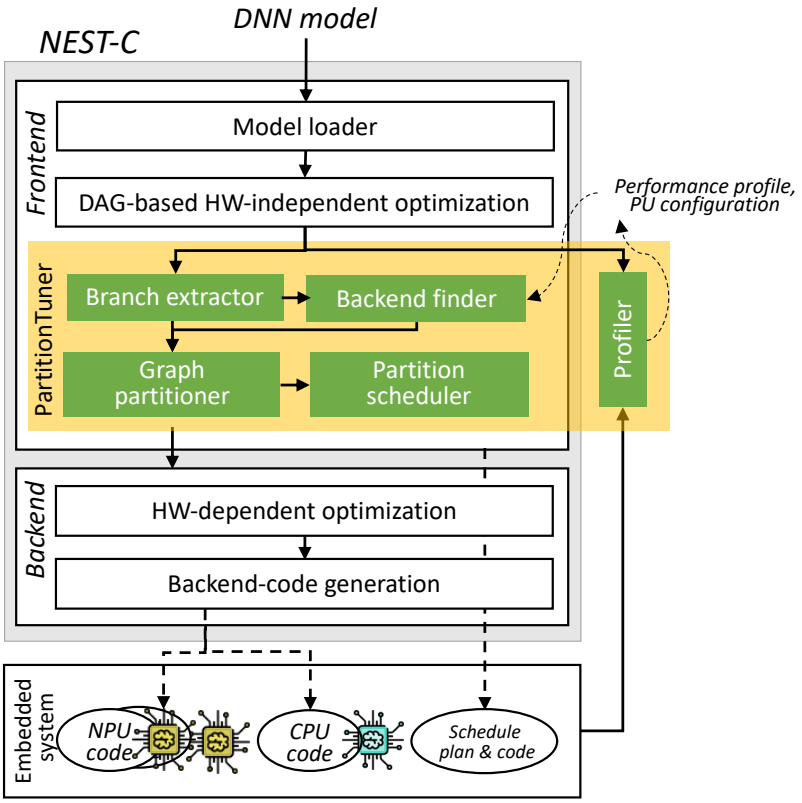
버전	일자	내역	작성자	승인자
1.0	2022.07.21	Version 1.0 작성	유미선	김태호
1.1	2022.12.20	파티셔너 파라미터 수정	유미선	

목차

1. 개요

NEST-C의 일부인 파티션 튜너는 NEST-C가 이종 프로세싱 유닛 (PU)을 동시에 사용하는 병렬 코드를 생성할 수 있도록 지원하는 도구이다. 파티션 튜너는 NEST-C의 계산 그래프(Graph IR)을 입력으로 받아 이종 PU에 할당 할 서브 그래프들로 나누고 각 서브 그래프들을 타겟에서 실행 가능한 머신 코드로 변환한다.

[그림 1]은 파티션 튜너를 구성하는 모듈을 보여준다.



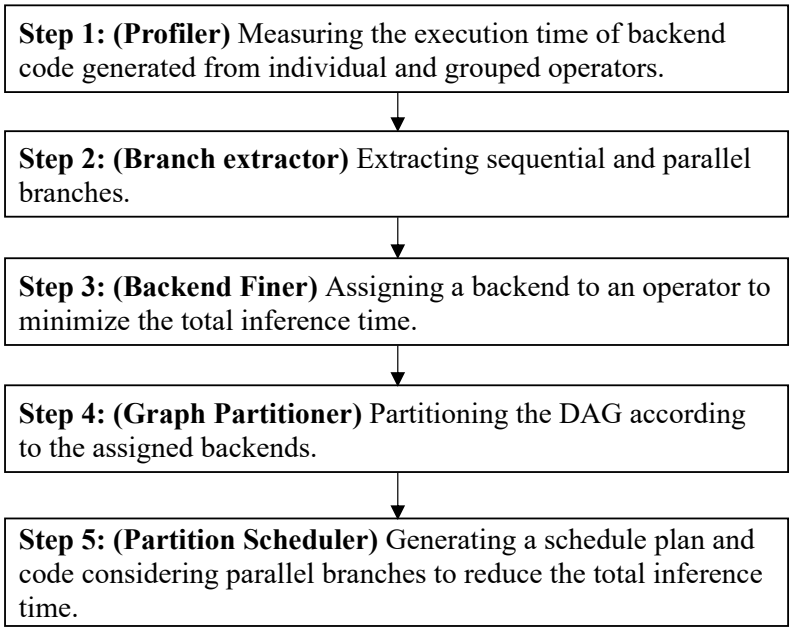
[그림 1] 파티션 튜너 구성 모듈

파티션 튜너는 branch extractor, backend finder, graph partitioner, partition scheduler, profiler로 이루어져 있다.

본 문서에서는 파티션 튜너를 사용하고자 하는 사용자를 위하여 모델로부터 최적 파티션을 자동으로 찾아내고, 다중 EVTA용 코드를 생성하는 방법에 대해 설명한다.

2. 최적 파티션 플랜 생성 방법

[그림 2]는 파티션 튜너의 실행 흐름이다. 최적 파티션은 입력 모델로부터 생성된 계산 그래프의 각 노드와 에지의 수행 시간, 즉, 텐서 연산자의 실행시간과 데이터 송수신에 걸리는 시간을 기반 (데이터 송수신 시간)을 기반으로 수행된다.
연산자 실행 시간 및 데이터 송수신 시간이 구해지면, 파티션 튜너는 해당 값들을 이용하여 어떠한 연산이 어느 PU에서 실행될 때 모델의 총 추론 시간이 최소가 될 수 있는지를 계산하여 그래프를 파티셔닝한다. 또한, 각 파티션들의 데이터 의존 관계 분석을 통해 순차적으로 실행되어야 할지 병렬로 실행 가능한지 결정한다.



[그림 2] 파티션 튜너의 실행 흐름

2.1 성능 프로파일 결과 생성

연산 실행 및 데이터 송수신 시간을 포함하는 성능 프로파일을 생성하기 위해서는 파티션 튜너를 이용하여 사용자가 다음과 같은 과정을 수행하여야 한다.

1) 각 PU들 상에서의 연산자 성능 프로파일링을 위한 파티션 플랜 생성

```
-model-input=data,float,[1,224,224,3] -m=/home/msyu/Dropbox/Development/Demo/mxnet_exported_resnet18.onnx -nest-log-partition -load-device-configs=/home/msyu/CLionProjects/nest_partition_etrij/tests/runtime_test/CPUEVTATestDeviceConfigs.yaml -profile-path="/home/msyu/Dropbox/Development/TestData/configs/resnet18_partition_perform_profile" -exe-type=1
```

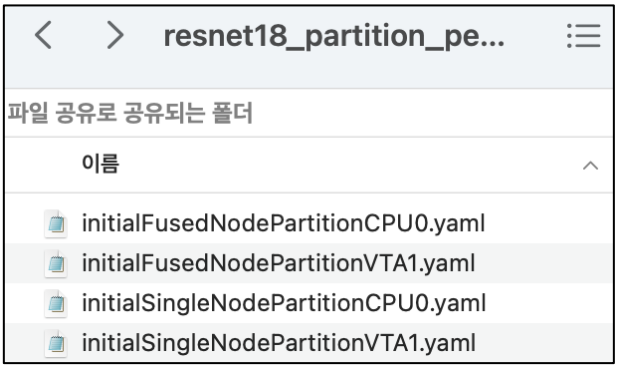
[그림 3] 프로파일링을 위한 파티션 플랜 생성 명령어

위 명령어를 수행하면 '-load-devide-configs' 옵션에 명시 된 PU들 상에서 입력 모델의 성능 프로파일링을 수행하는데 필요한 파티션 플랜이 생성된다. 프로파일링은 단일 연산자의 성능 뿐 아니라 연산자 그룹을 연속적으로 수행했을 때의 성능도 측정하는데, 그룹에 속하는 연산자의 종류는 '-profile-path'로 명시된 폴더 내의 'profileConfigs.yaml' 파일 내에 명시되어 있어야 한다. [그림 4]는 convolution과 relu 연산을 그룹으로 명시한 profileConfigs.yaml 파일의 예이다.

```
---
  name: Profile1
  fuseOperators: |
    Convolution-Relu
...
```

[그림 4] profileConfigs.yaml 예제

[그림 3]의 명령어를 수행하면 프로파일 폴더 아래에 다음과 같은 yaml 파일들이 생성된다. 이 파일들은 프로파일링 코드 생성에 필요한 연산자 분할 및 코드 생성에 사용 될 백엔드 정보를 포함하고 있는 파티션 플랜 파일들이다.



2) 프로파일링 코드 생성

이전 단계에서 생성한 파티션 플랜들을 이용하여 실제 타겟 시스템에서 실행할 프로파일링 코드를 생성한다. [그림 5]는 이전 단계에서 생성한 프로파일용 파티션 플랜을 이용하여 프

로파일링 코드를 생성하는 명령어이다.

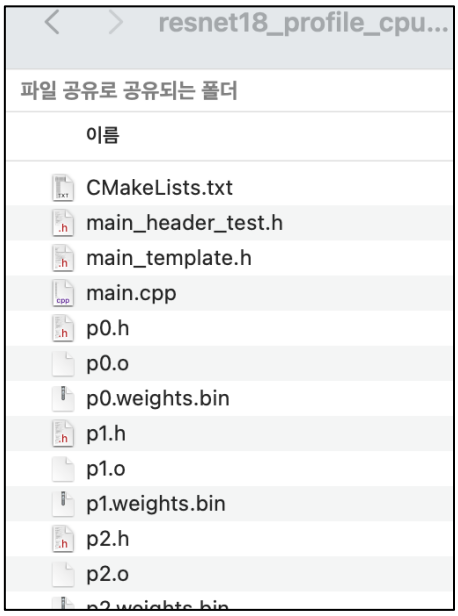
```
-model-input=data,float,[1,224,224,3] -m=/home/msyu/Dropbox/Development/Demo/mxnet_exported_resnet18.onnx -nest-log-partition -load-device-configs=/home/msyu/CLionProjects/nest_partition_etrij/tests/runtime_test/CPUEVTATestDeviceConfigs.yaml -exe-type=0 -target=aarch64 -mcpu=cortex-a53 -backend=CPU -emit-bundle=resnet18_profile_cpu_single -bundle-api=dynamic -partition-plan=/home/msyu/Dropbox/Development/TestData/configs/resnet18_partition_perform_profile/initialSingleNodePartitionCPU0.yaml -profile-mode=1
```

[그림 5] 프로파일링 코드 생성 명령어

위 명령어에서 '-emit-bundle' 옵션의 값은 코드가 저장될 폴더 이름이며 '-partition-plan' 옵션값은 파티션 플랜 파일의 경로이다. 프로파일링 코드를 생성하기 위해서는 '-profile-mode'의 값을 반드시 1로 설정해 주어야 한다.

3) 프로파일링 코드 실행을 통한 성능 프로파일 생성

위 명령어를 실행하면 아래와 같이 main.cpp, CMakeLists.txt, 기타 오브젝트 파일 및 소스코드들이 생성된다 ([그림 6]).



[그림 6] 프로파일링 코드 예제

코드가 생성된 폴더에 NEST-C에서 제공하는 main_header_test.h 및 main_template.h 파일을 복사하고, 해당 폴더를 프로파일링을 수행하고자 하는 타겟 시스템 내의 NEST-C의

소스 디렉토리에 복사하고 빌드 디렉토리에서 make 명령어를 사용하여 빌드하면 실행 파일이 생성된다. 생성된 실행파일을 임의의 입력을 주어 [그림 7]과 같이 실행시킨다.

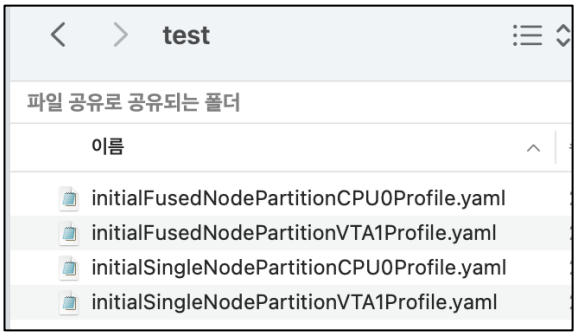
```
root@pynq:~/nest_multivta_zcu_exp/build_release/vta/bundles/resnet18/resnet18_profile_cpu_single#
./demoExeFileName21555 cat_285.png
```

[그림 7] 프로파일링 수행 예제

위 명령어를 실행시면 현재 디렉토리 안에 연산의 수행시간과 데이터 송수신 시간을 저장한 2개의 .yaml 파일 (성능 프로파일)이 생성된다.

4) 프로파일링 결과를 폴더에 저장

이전 단계에서 생성된 성능 프로파일들을 파티션 튜너를 실행하는 컴퓨터의 특정 폴더에 모아서 복사한다.



[그림 8] 프로파일링 결과 파일 예제

2.2 배치코드 생성을 위한 파티션 플랜 생성

파티션 튜너 실행 시에 '-exe-type=2'로 설정하고, '-profile-path'에 프로파일링 결과를 저장한 폴더를 지정하면 멀티 EVTA를 사용하는 파티션 플랜을 생성할 수 있다. 사용 할 EVTA의 개수는 -evta-num으로 설정한다. [그림 9]는 4개의 EVTA를 사용하는 코드를 생성하는 명령어 예제이다.

''

```
-model-input=data,float,[1,224,224,3] -m=/home/msyu/Dropbox/Develop-
ment/Demo/mxnet_exported_resnet18.onnx -nest-log-partition -load-device-con-
```

```
figs=/home/msyu/CLionProjects/nest_partition_etrij/tests/runtime_test/CPUEVTATestDeviceConfigs.yaml -profile-path="/home/msyu/Dropbox/Development/TestData/configs/resnet18_partition_perform_profile" -exe-type=2 -evta-num=4
```

[그림 9] 멀티 EVTA를 사용하는 최적 파티션 플랜 생성 명령어

위 명령어를 수행하면 4개의 EVTA를 사용하는 최적 파티션 플랜이 생성된다. 현재 파티션 튜너는 EVTA를 최대 4개까지 사용하도록 설정되어 있으며, 파티션 플랜 내에서 각 EVTA는 'VTA-0', 'VTA-1', 'VTA-2', 'VTA-3'로 참조된다. 또한, 서로 다른 EVTA에서 동시에 실행되는 파티션들은 'parallelPartition' 키워드를 이용하여 명시한다. [그림 10]는 파티션 튜너에 의해 자동 생성된 멀티 EVTA용 최적 파티션 플랜의 예를 보여준다. (1)은 각 파티션에 매칭 된 백엔드 이름 부분이며, (2)는 병렬 수행 되는 파티션을 표기한 부분을 보여준다.

(1) 멀티 VTA 표기

```
backendNames: Relay:VTA:Relay:VTA:Relay:VTA-0:VTA-1:Relay:VTA:Relay:VTA-0:VTA-1:Re
```

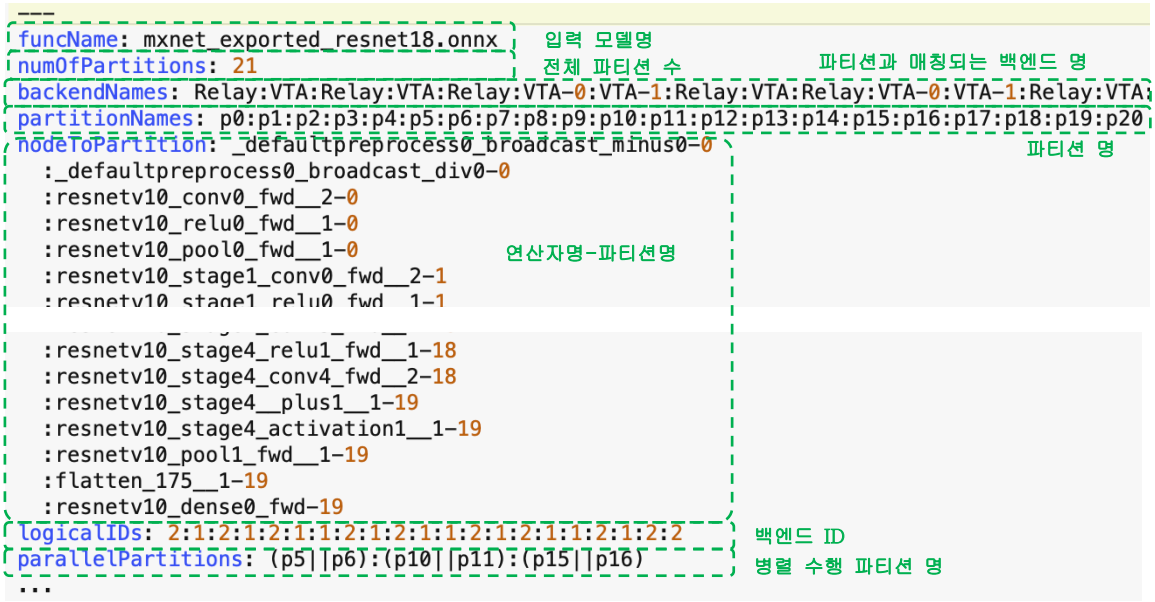
(2) 병렬 실행 파티션 표기

```
parallelPartitions: (p5||p6):(p10||p11):(p15||p16)
```

[그림 10] ResNet18 용 최적 파티션 플랜의 예

2.3 수동으로 멀티 EVTA를 활용하는 파티션 플랜 생성 방법

파티션 튜너는 파티션 플랜을 yaml 파일로 출력하고, 출력된 파티션 플랜을 사용하여 타겟에서 컴파일 가능한 코드를 생성한다. 그러므로 yaml 형태의 파티션 플랜을 직접 수정하여 원하는 형태로 파티셔닝을 수행 하여 코드를 생성해 볼 수 있다. 파티션 튜너가 사용하는 파티션 플랜이 포함해야 하는 정보는 아래와 같다.



[그림 11] 파티션이 포함하는 정보

3. 파티션 플랜으로부터 실행 코드를 생성하는 방법

파티션 튜너에서 생성되는 파티션 플랜은 파티션 튜너에 의해 다시 코드로 변환될 수 있다. 파티션 튜너의 옵션인 '-exe-type'을 '0'으로 설정하고 '-partition-plan'에 파티션 플랜 파일의 경로를 설정하여 파티션 튜너를 실행하면 타겟 시스템에서 빌드 가능한 코드가 생성된다. [그림 12]는 멀티 EVTA용 최적 파티션 플랜 (NESTMinCostOptimalPlan-multiVTA.yaml)으로부터 코드를 생성하는 커맨드라인 명령어 예제이다.

```
-model-input=data,float,[1,224,224,3]                    -m=/home/msyu/Dropbox/Develop-
```

```
ment/Demo/mxnet_exported_resnet18.onnx -nest-log-partition -load-device-con-
figs=/home/msyu/CLionProjects/nest_partition_etrij/tests/runtime_test/CPUEVTATestDeviceCon-
figs.yaml -exe-type=0 -target=aarch64 -mcpu=cortex-a53 -emit-bundle=resnet18_mincost_op-
timal_multivta -bundle-api=dynamic -partition-plan=/home/msyu/Dropbox/Development/Test-
Data/configs/resnet18_partition_perform_profile/NESTMinCostOptimalPlan-multiVTA.yaml -pro-
file-mode=0 -load-profile=/home/msyu/Dropbox/Development/TestData/configs/resnet18_parti-
tion_perform_profile/mxnet_exported_resnet18_calib_1.yaml -relay-target=aarch64 -relay-ex-
port-option=aarch64 -relay-opt-level=3
```

[그림 12] 파티션 플랜으로부터 코드를 생성하는 명령어

위 커맨드 라인 명령을 통해 생성된 코드를 타겟 시스템으로 옮겨 빌드하면 실행 프로그램이 생성된다. 타겟으로 옮기고 빌드하는 절차는 2.1절의 3)-4)에서 설명된 프로파일링 코드 빌드 및 실행 절차 동일하다.