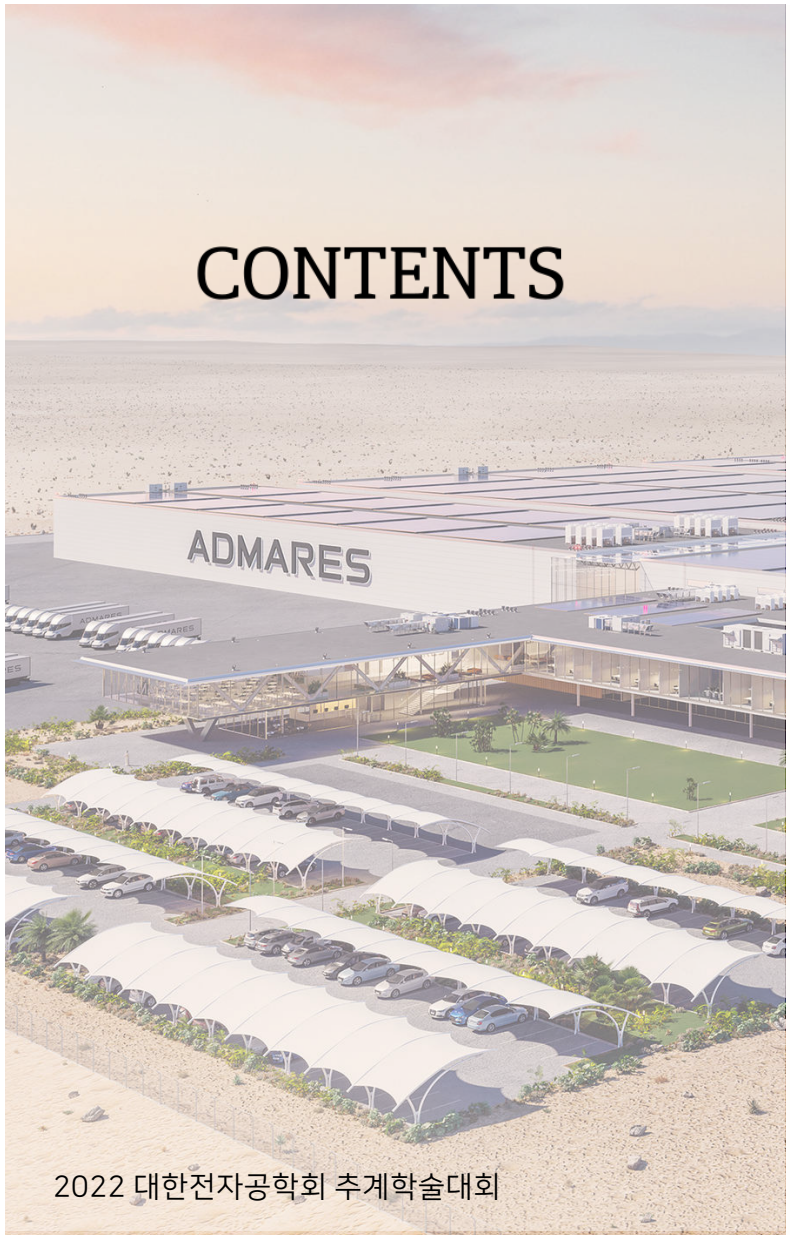
단일 ISA 이기종 멀티 코어 구조를 위한 프로파일 기반 ArmCL 최적 스케줄 탐색

*Joo Hyoung Cha, **Jubin Lee, ***Yongin Kwon

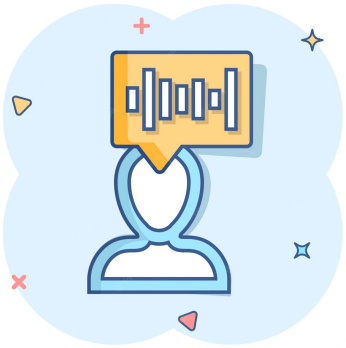
*Dong-Eui University
**University of Science and Technology
***Electronics and Telecommunications Research Institute

2022 대한전자공학회 추계학술대회



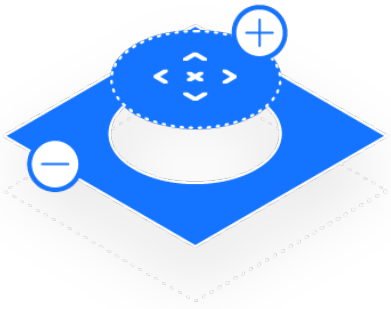
01 연구 목적

영상 속 객체 검출 및 분류



음성 분류 / 인식

이미지 속 글자 검출 / 번역



전경과 배경 분리

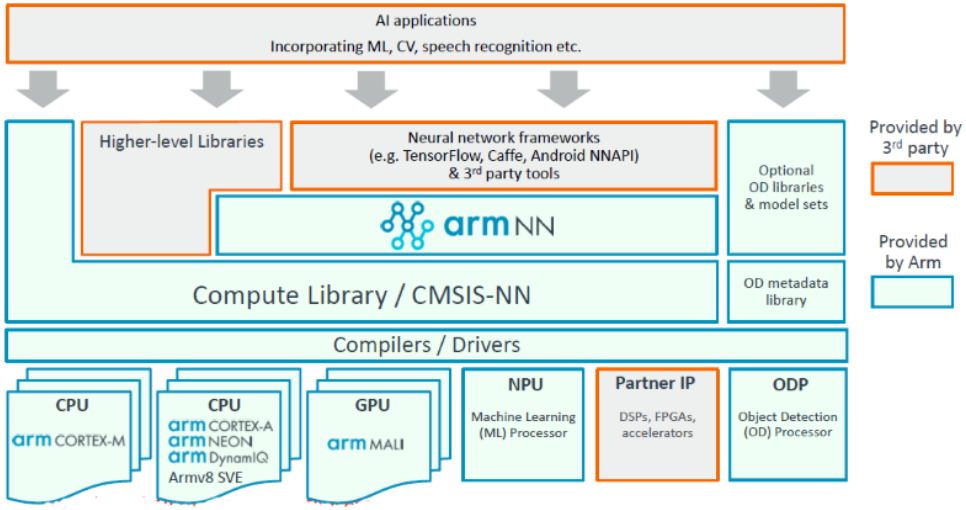
임베디드 시스템의 성능 향상으로 인하여
인공지능을 활용한 웹 / 어플리케이션이 증가함.

01 연구 목적

ARM, 딥 러닝 가속 라이브러리

- Arm의 SIMD를 활용하여 딥 러닝 가속 라이브러리를 제공함.
  - ✓ ArmCL : Arm에서 제공하는 딥러닝 라이브러리
  - ✓ SIMD : Single Instruction Multiple Data
    하나의 명령어로 병렬 연산을 수행함.

- ArmCL이 적용 된 사례
  - ✓ OpenVINO : Intel의 딥러닝 프레임워크
  - ✓ Tensorflow : Google의 딥러닝 학습 프레임워크
  - ✓ NCNN : Tencent의 신경망 추론 플랫폼



Arm의 CPU에서 딥 러닝 연산을 위해 ArmCL을 활용함.

02 문제 인식

테스트 환경

- **인공지능 모델 명 : AlexNet**
- **배치 사이즈 : 100**
- **CPU 코어 수 정보 (SoC 명 : RK3399Pro)**
  - ✓ Big 코어 수 : 2
  - ✓ Little 코어 수 : 4
- **테스트 관련**
  - ✓ 증명을 위해 특정 코어 비활성화 하였음.
  - ✓ 성능 측정은 내장 고해상도 시계를 활용하였음.

- **Big 연산 성능 : 12.8409 초 (기본 설정)**

```
top - 12:32:13 up 13 days, 21:16,  5 users,  load average: 0.17, 0.29, 0.26
Tasks: 222 total,   2 running, 220 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.7 us,  0.7 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.7 us,  1.0 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.3 us,  0.7 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 99.3 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3845.3 total,    103.1 free,   1418.9 used,   2323.4 buff/cache
MiB Swap:   2048.0 total,   1924.2 free,    123.8 used.   2118.7 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
13915 linaro    20   0 1066244   1.0g   8328 R 199.7  26.8   0:18.87 graph_alexnet
13152 linaro    20   0    8004   3380   2576 R   1.3   0.1   0:07.47 top
```

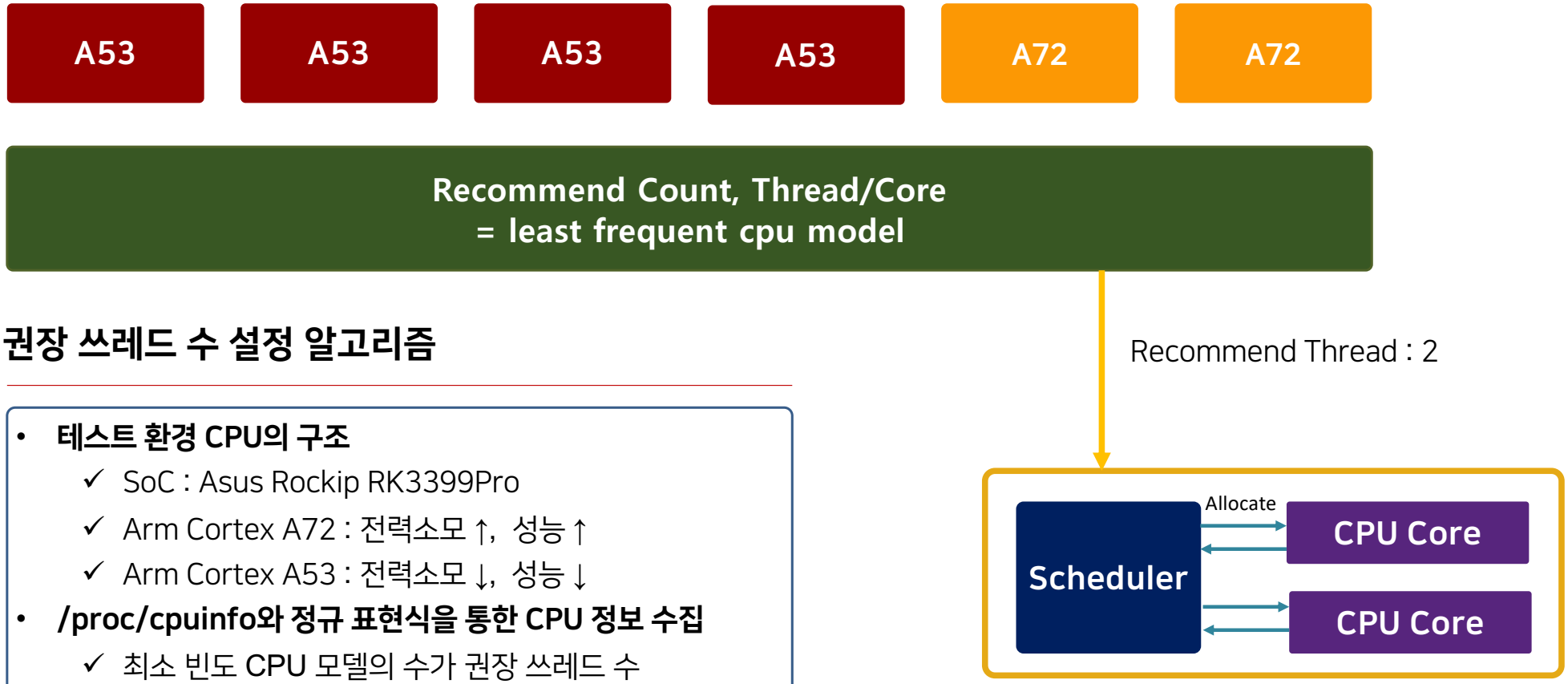
- **Little 코어 연산 성능 : 16.0998 초**

```
top - 12:35:37 up 13 days, 21:20,  5 users,  load average: 1.02, 0.62, 0.40
Tasks: 223 total,   2 running, 221 sleeping,   0 stopped,   0 zombie
%Cpu0  : 99.3 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 95.0 us,  0.7 sy,  0.0 ni,  4.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3845.3 total,     21.5 free,   1500.3 used,   2323.5 buff/cache
MiB Swap:   2048.0 total,   1924.2 free,    123.8 used.   2037.2 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
14118 linaro    20   0 1166232   1.1g   8136 S 393.1  28.9   0:53.47 graph_alexnet
13152 linaro    20   0    8004   3380   2576 R   0.7   0.1   0:09.64 top
    1 root      20   0  165760   6628   4452 S   0.3   0.2 195:13.42 systemd
  719 root      20   0   95132   2256   2020 S   0.3   0.1  12:24.29 npu_transfer_pr
```

- **Big + Little 코어 연산 성능 : 9.8625 초**

```
top - 12:33:20 up 13 days, 21:18,  5 users,  load average: 1.73, 0.67, 0.40
Tasks: 223 total,   3 running, 220 sleeping,   0 stopped,   0 zombie
%Cpu0  : 58.9 us,  0.3 sy,  0.0 ni, 40.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 66.4 us,  0.3 sy,  0.0 ni, 33.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 63.8 us,  0.7 sy,  0.0 ni, 35.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 50.0 us,  0.3 sy,  0.0 ni, 49.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 92.1 us,  0.7 sy,  0.0 ni,  7.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 94.4 us,  0.3 sy,  0.0 ni,  5.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3845.3 total,    102.3 free,   1419.4 used,   2323.7 buff/cache
MiB Swap:   2048.0 total,   1924.2 free,    123.8 used.   2118.2 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
13986 linaro    20   0 1099028   1.0g   8288 R 425.2  26.8   0:42.77 graph_alexnet
13152 linaro    20   0    8004   3380   2576 R   0.7   0.1   0:08.22 top
```

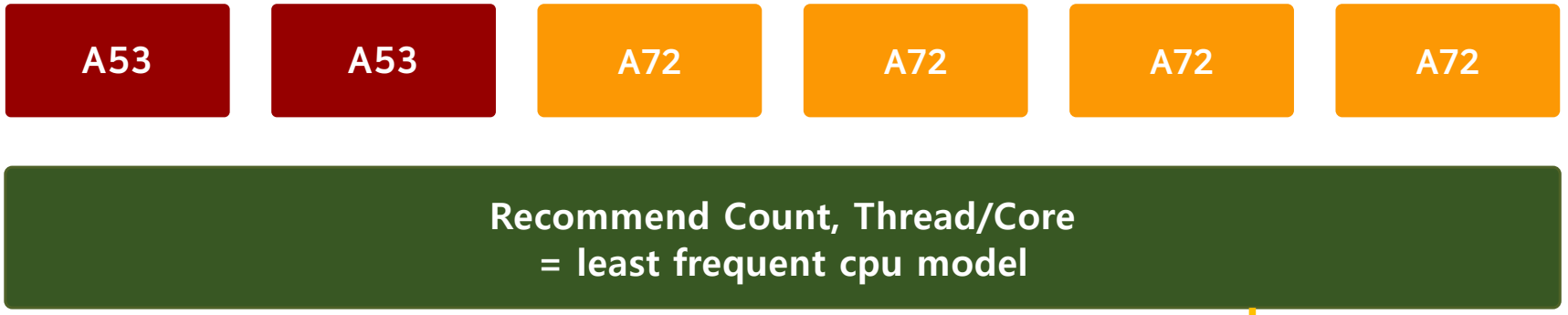
02 문제 인식: 스케줄링



권장 쓰레드 수 설정 알고리즘

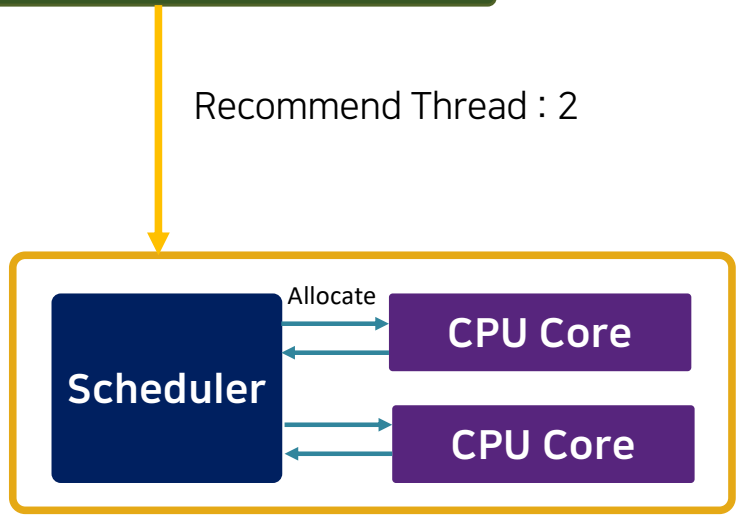
- **테스트 환경 CPU의 구조**
  - ✓ SoC : Asus Rockip RK3399Pro
  - ✓ Arm Cortex A72 : 전력소모 ↑, 성능 ↑
  - ✓ Arm Cortex A53 : 전력소모 ↓, 성능 ↓
- **/proc/cpuinfo와 정규 표현식을 통한 CPU 정보 수집**
  - ✓ 최소 빈도 CPU 모델의 수가 권장 쓰레드 수
  - ✓ A72를 활성화 한다면 권장 쓰레드 수는 2

02 문제 인식: 스케줄링



권장 쓰레드 수 설정 알고리즘

- **테스트 환경 CPU의 구조**
  - ✓ Arm Cortex A72 : 전력소모 ↑, 성능 ↑
  - ✓ Arm Cortex A53 : 전력소모 ↓, 성능 ↓
- **예외의 케이스로 역전 된다면?**
  - ✓ 최소 빈도 CPU 모델의 수가 권장 쓰레드 수
  - ✓ A53이 최소 빈도 이므로 결과 값은 2
  - ✓ 결과적으로 해당 문제를 해결한다면 성능 향상됨



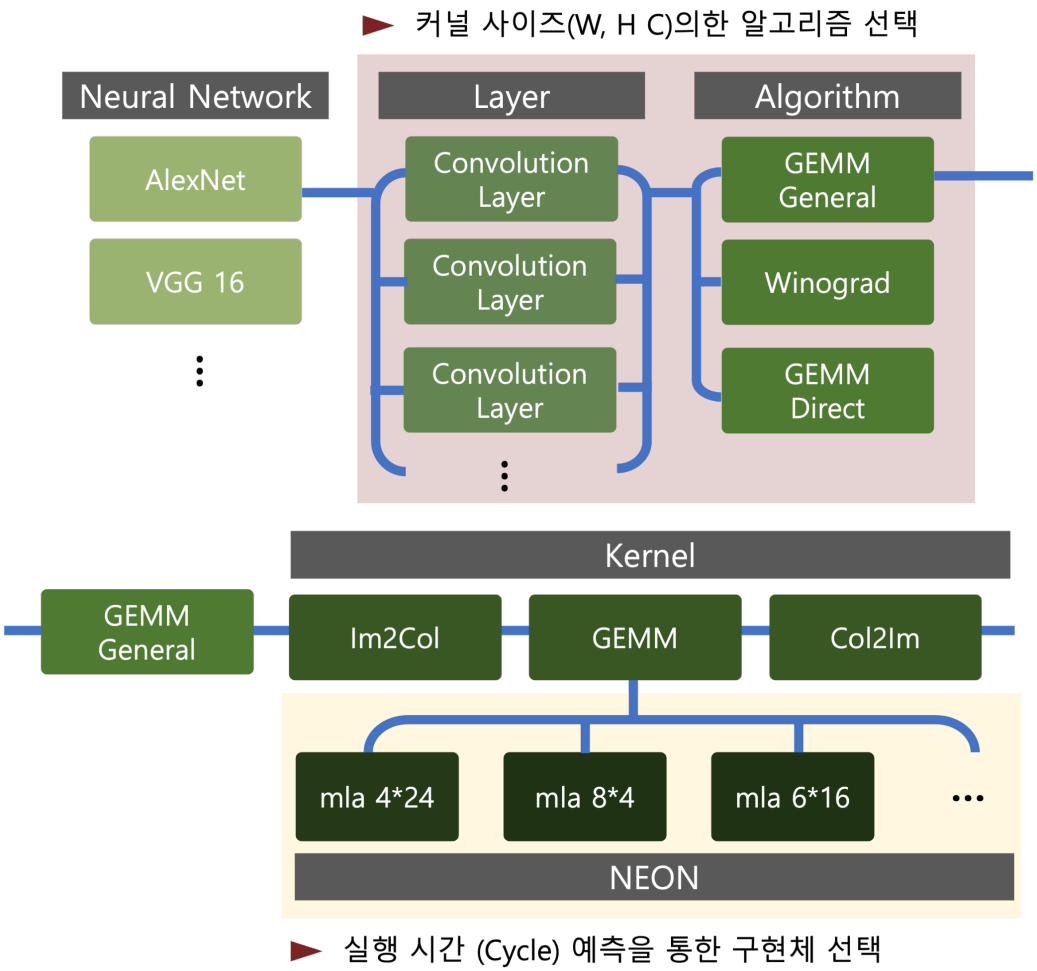
02 문제 인식: 내부 ArmCL 구조

Layer: Abstract Node

- **AlexNet**
  - ✓ Fully Connected Layer
  - ✓ Convolution Layer
  - ✓ Max-Pooling Layer
  - ✓ Concatenate Layer

Algorithm: Implement Node

- **Convolution Layer**
  - ✓ GEMM Convolution Layer
  - ✓ Winograd Convolution Layer
  - ✓ Naïve Convolution Layer
  - ✓ Fast Fourier Transform Convolution Layer
- **Max-Pooling Layer 그 외 …**



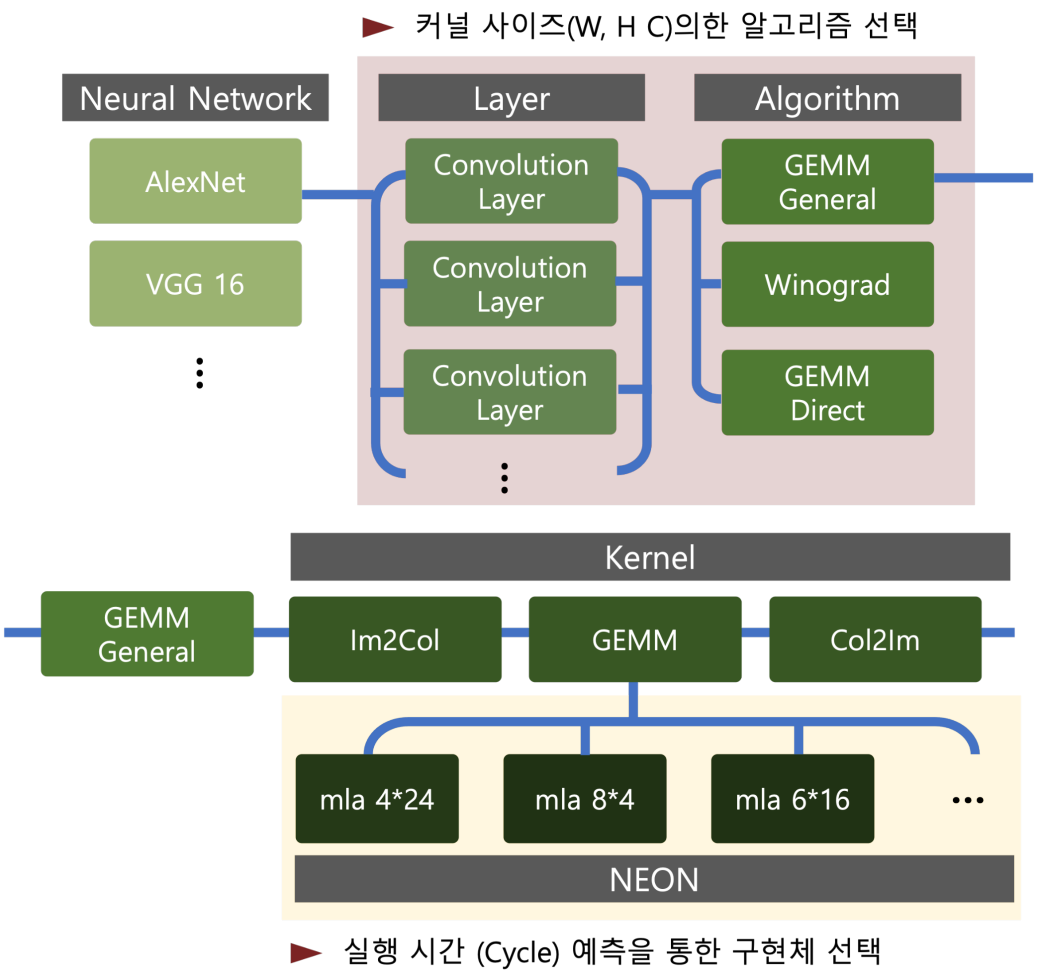
02 문제 인식: 내부 ArmCL 구조

Kernel: Raw Ops

- **GEMM Convolution Layer**
  - ✓ Im2Col Kernel
  - ✓ GEneral Matrix to Matrix(GEMM) Kernel
  - ✓ Col2Im Kernel
- **ArmCL에서 텐서를 제어하는 최소 논리 연산자**
  - ✓ 모든 알고리즘은 1개 이상의 Kernel이 존재함.

Assembly for ARM NEON

- **GEneral Matrix to Matrix(GEMM) Kernel의 구현체**
  - ✓ mla 4x24, mla 8x4, mla 6x16
  - ✓ sgemm 8x6, sgemm 8x12
- **논리적인 개념에서 연산을 수행하는 단계**
  - ✓ 모든 Kernel은 1개 이상의 구현체가 존재함.

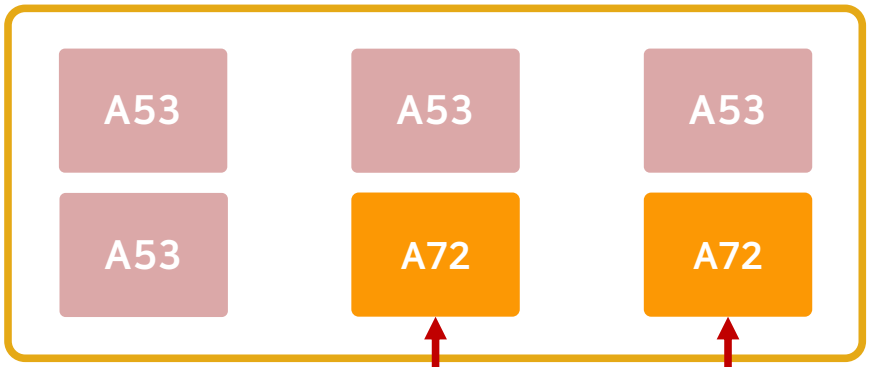


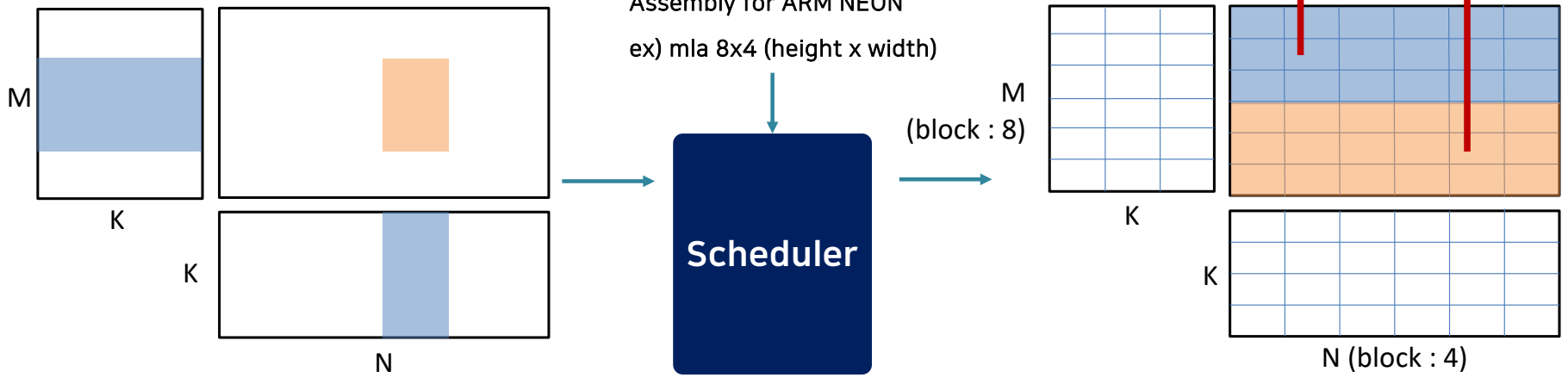
03 실험 수행 과정: 작업량 분배

Partition Scheduling Algorithm

- **GEMM Kernel의 파티셔닝 과정**
  - ✓ Kernel의 구현체에 따라 작업 사이즈가 변동됨.
  - ✓ 권장 쓰레드 수에 맞춰 코어 별 할당 함.
  - ✓ 쓰레드는 운영체제 커널의 스케줄링 알고리즘에 따라 특정 코어에 동작함
- **전체 작업 사이즈 계산 식**
  - ✓ $work\_size\_of\_kernel = \left\lceil \frac{M}{height} \right\rceil * \left\lceil \frac{N}{width} \right\rceil$

Allocate Workload: GEMM Kernel





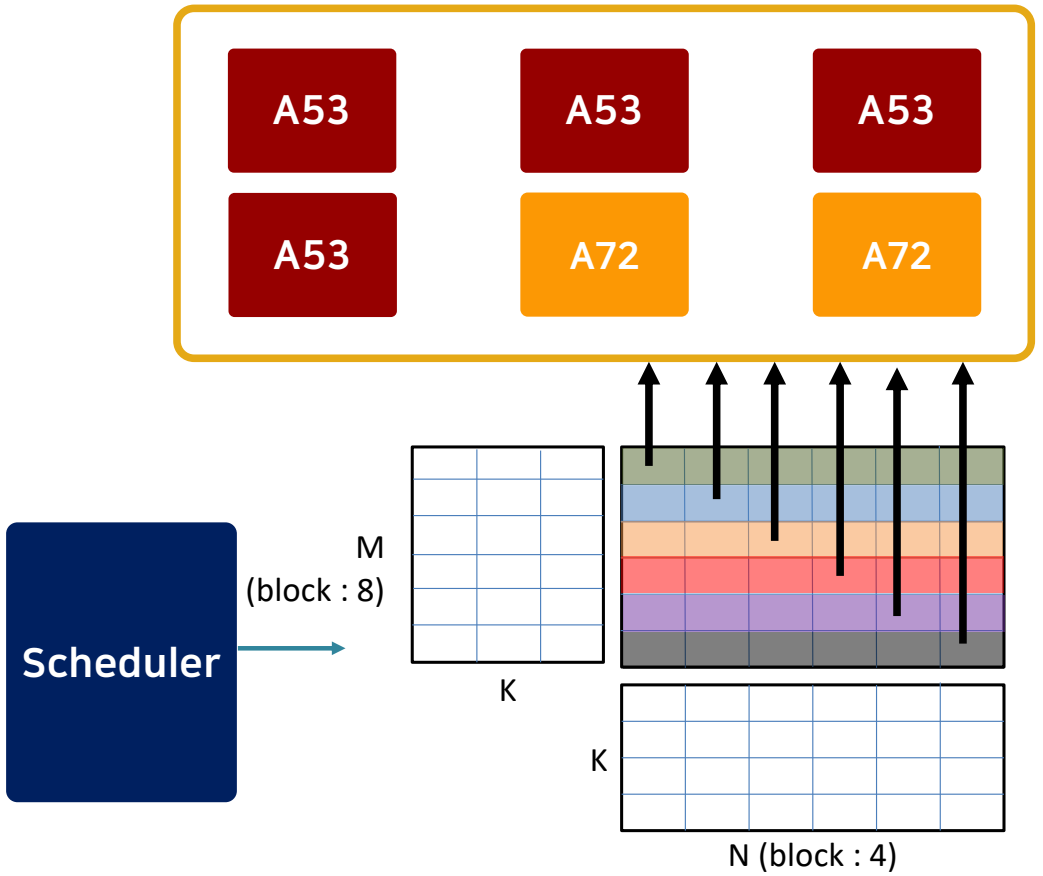
03 실험 수행 과정: 작업량 분배

Partition Scheduling Algorithm

- **코어 별 작업 량**
  - ✓ Kernel의 구현체에 따라 전체 작업 사이즈가 변동됨.
  - ✓ 전체 작업 량을 각 쓰레드에 분배함.
  - ✓ 쓰레드는 코어 선호도를 이용하여 지정함.
- **조작 변인**
  - ✓ 코어 별 작업 사이즈 제어
- **통제 변인**
  - ✓ 코어 별 쓰레드 수 : 1
  - ✓ 캐시 메모리
  - ✓ 코어 클럭 속도

Big Core	Little Core
1.8 Ghz	1.416 Ghz

Allocate Workload: GEMM Kernel



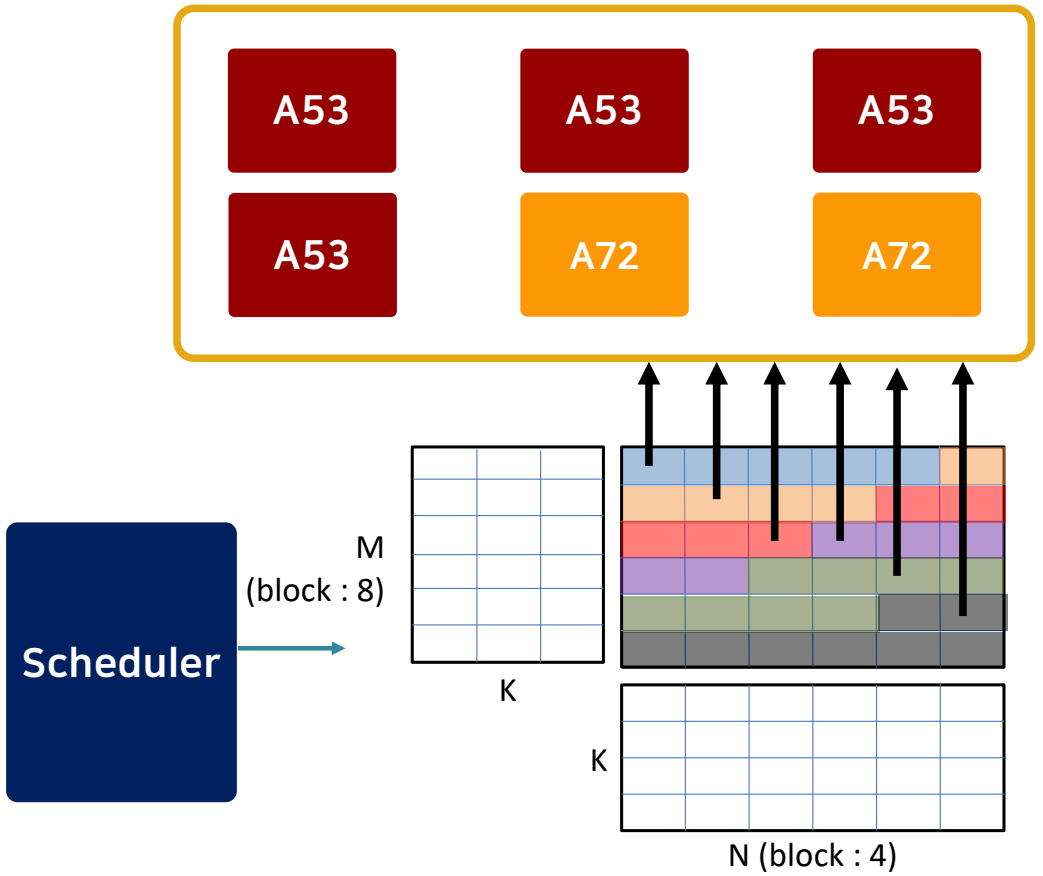
03 실험 수행 과정: 작업량 분배

Partition Scheduling Algorithm

- **코어 별 작업 량**
  - ✓ Kernel의 구현체에 따라 전체 작업 사이즈가 변동됨.
  - ✓ 전체 작업 량을 각 쓰레드에 분배함.
  - ✓ 쓰레드는 코어 선호도를 이용하여 지정함.
- **조작 변인**
  - ✓ 코어 별 작업 사이즈 제어
- **통제 변인**
  - ✓ 코어 별 쓰레드 수 : 1
  - ✓ 캐시 메모리
  - ✓ 코어 클럭 속도

Big Core	Little Core
1.8 Ghz	1.416 Ghz

Allocate Workload: GEMM Kernel



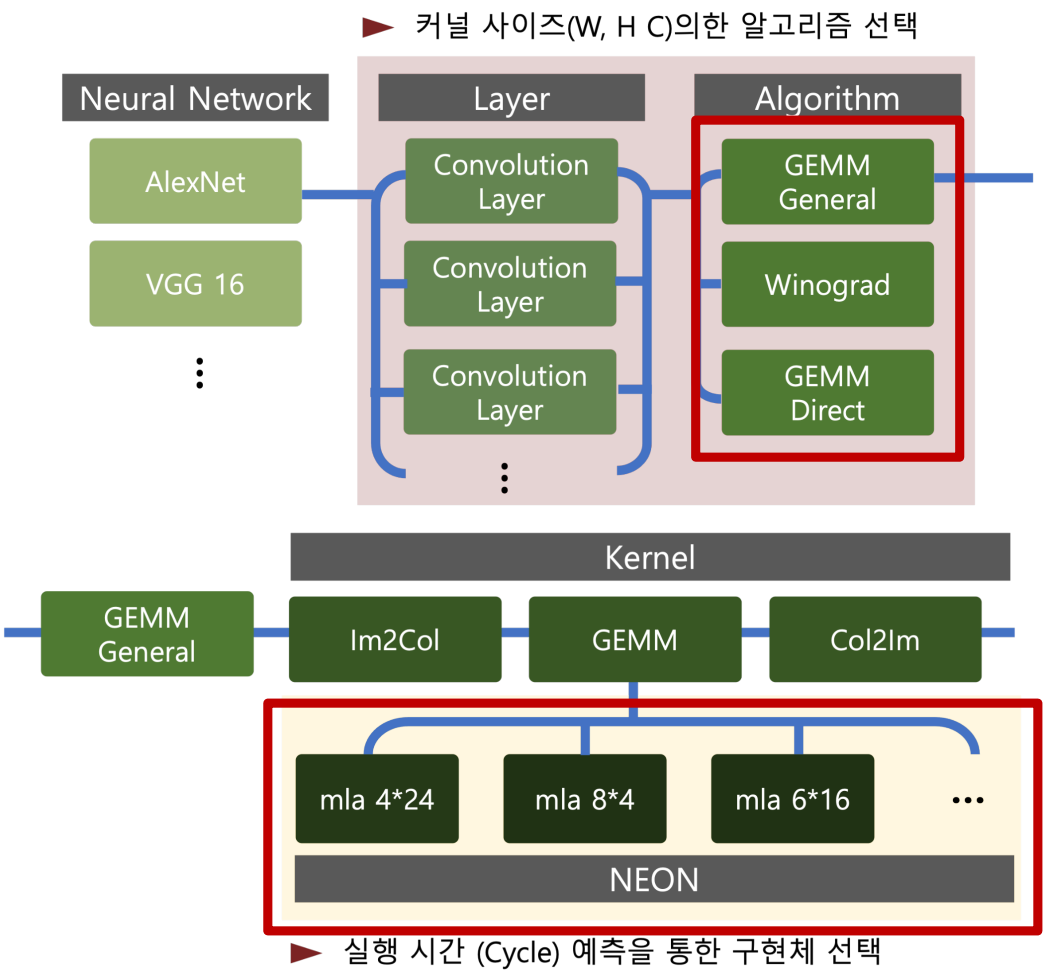
03 실험 수행 과정: 이기종에 적합한 알고리즘 선택

Optimize: Convolution Layer

- **이기종 코어 연산에 적합한 연산 알고리즘**
  - ✓ 컨볼루션 연산은 3가지의 알고리즘이 존재함.
  - ✓ 3가지의 알고리즘 모두 GEMM 연산이 필수적임.
  - ✓ GEMM Kernel은 5가지의 구현체가 존재함.

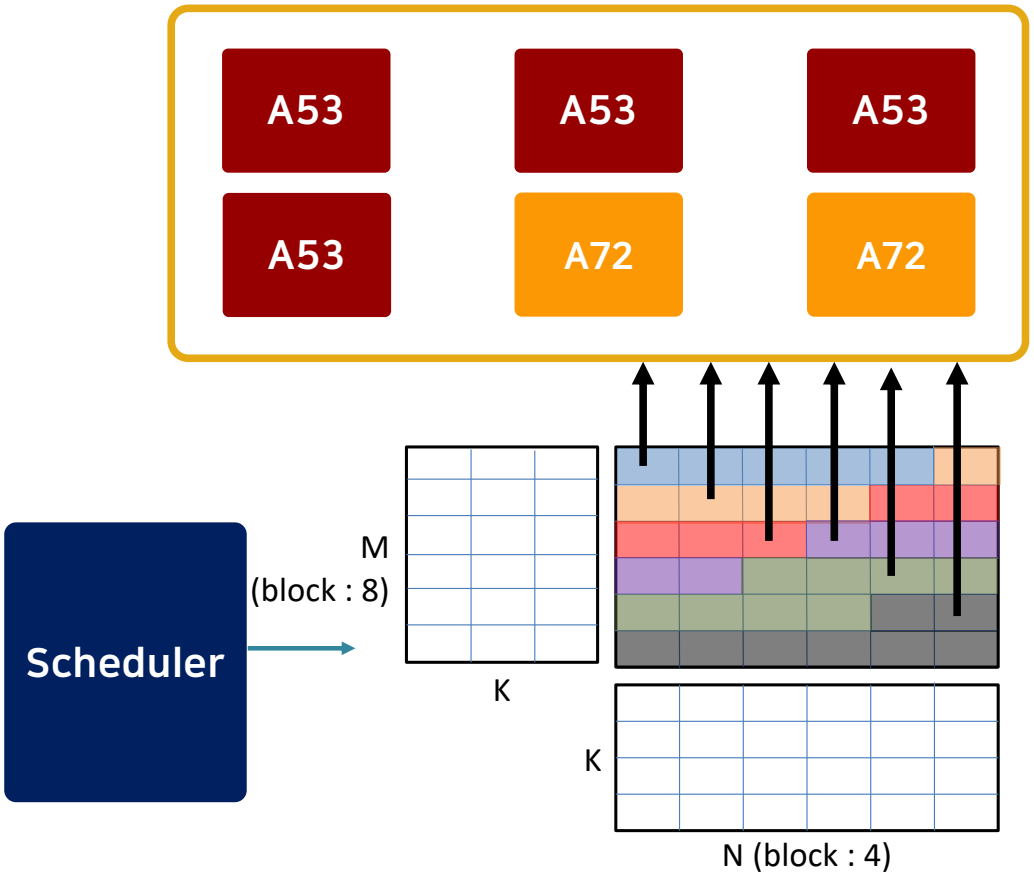
- **컨볼루션 레이어 최적화 관련**
  - ✓ 15가지의 경우를 모두 수행하여 최적 해를 탐색함.

컨볼루션 알고리즘	GEMM Kernel의 구현체
Winograd	Sgemm 8*6
Gemm General	Sgemm 8*12
	mla 8*4
Gemm Direct	mla 4*24
	mla 6*16

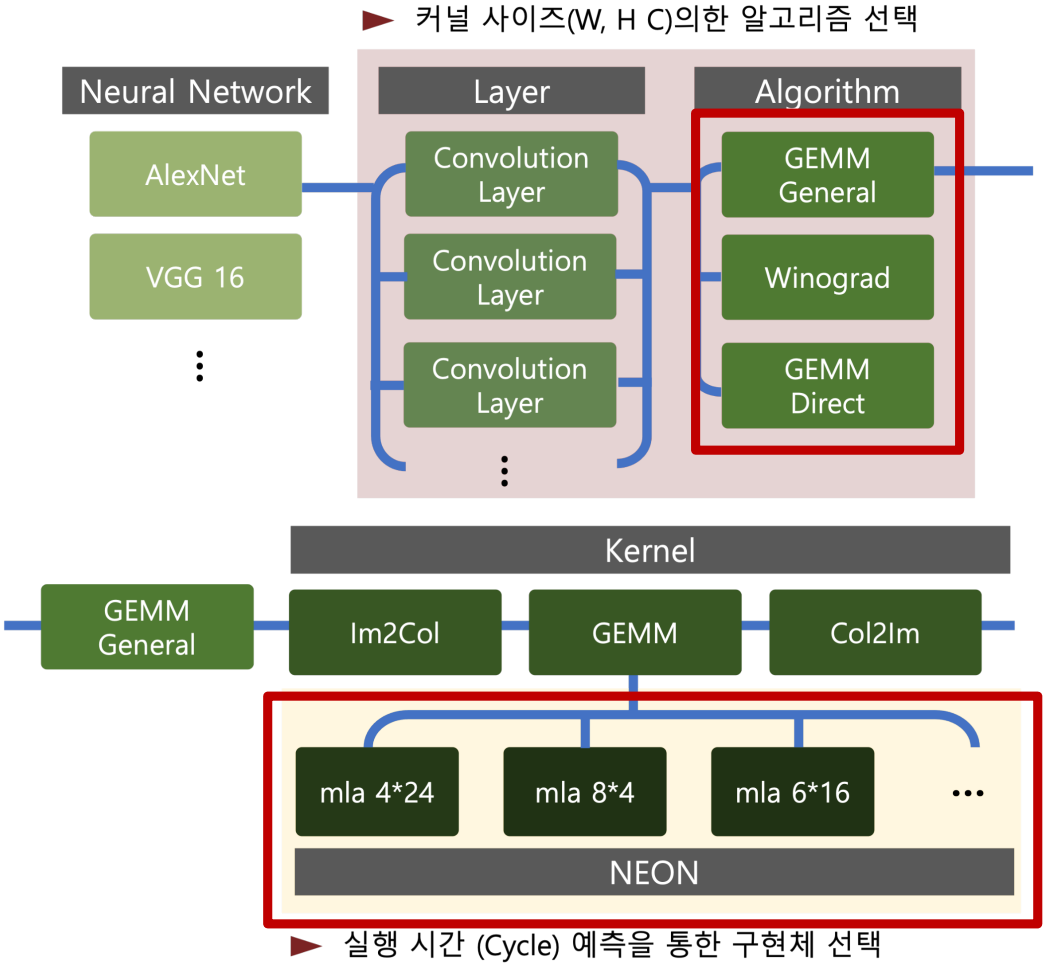


03 실험 수행 과정: 실험 정의

Allocate Workload: GEMM Kernel



► 커널 사이즈(W, H C)의한 알고리즘 선택

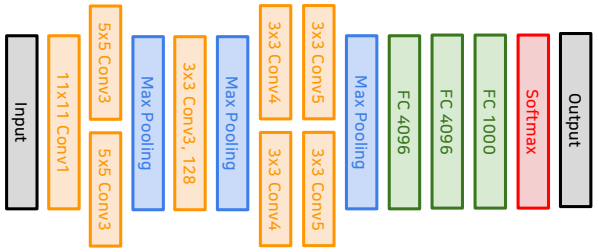


► 실행 시간 (Cycle) 예측을 통한 구현체 선택

04 실험 조건 정의

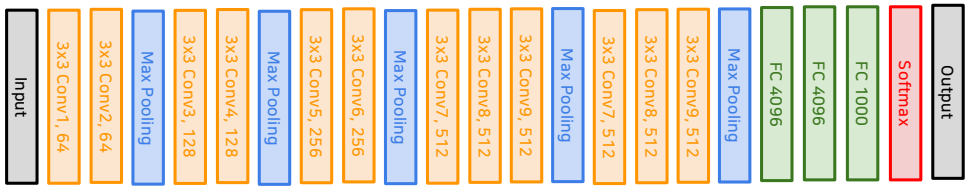
실험 모델 정의

1. AlexNet의 구조도 (ILSVRC 2012)



- AlexNet의 특징 (ILSVRC 2012)
  - ✓ 입력 사이즈 : 227x227x3
  - ✓ GPU 병렬 연산을 위해 병렬 구조로 설계 되었음.
  - ✓ 딥 러닝을 통해 최초로 이미지넷 대회에서 우승함.

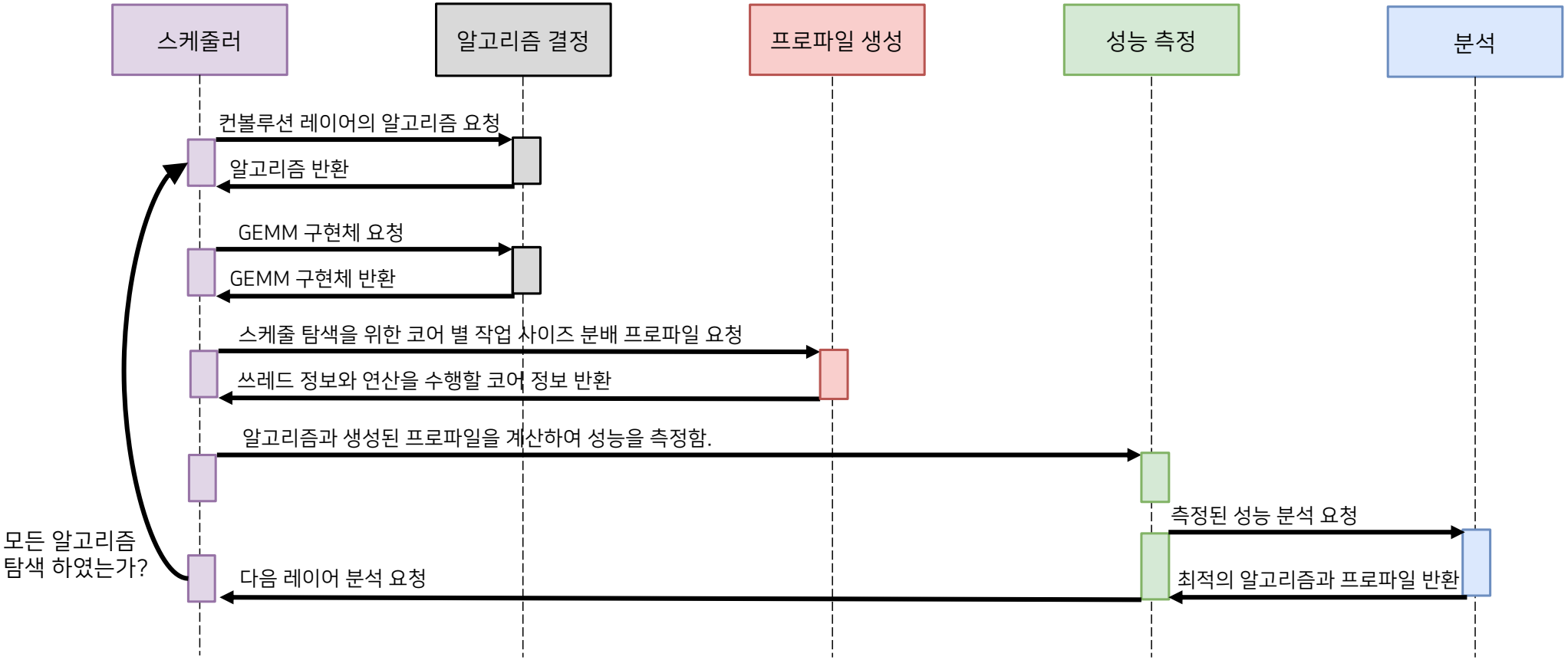
2. VGG 16의 구조도 (ILSVRC 2014)



- **VGG 16의 특징 (ILSVRC 2014)**
  - ✓ 입력 사이즈 : 224x224x3
  - ✓ 네트워크 깊이의 영향을 확인하기 위해 구현된 모델임.
  - ✓ 컨볼루션의 커널 필터 사이즈가 동일함

04 최적 스케줄 탐색 알고리즘

컨볼루션 레이어의 스케줄 탐색 알고리즘



04 최적 스케줄 탐색 알고리즘

프로파일 생성 알고리즘

- **설정 값**
  - ✓ 코어 선택 : { 'little(A53)', 'big(A72)', 'little+big' }
  - ✓ 코어 별 최소 작업량(C_{min}) : $\left\lceil \frac{work\_size\_of\_kernel}{20 * count(core)} \right\rceil$
  - ✓ 코어 별 최대 작업량 (C_{max}) : $\left\lceil \frac{work\_size\_of\_kernel}{5 * count(core)} \right\rceil$
  - ✓ 코어 별 작업 스텝 사이즈 (C_{step}): $\left\lceil \frac{work\_size\_of\_kernel}{100 * count(core)} \right\rceil$
- **전체 프로파일 생성시 발생하는 경우의 수**

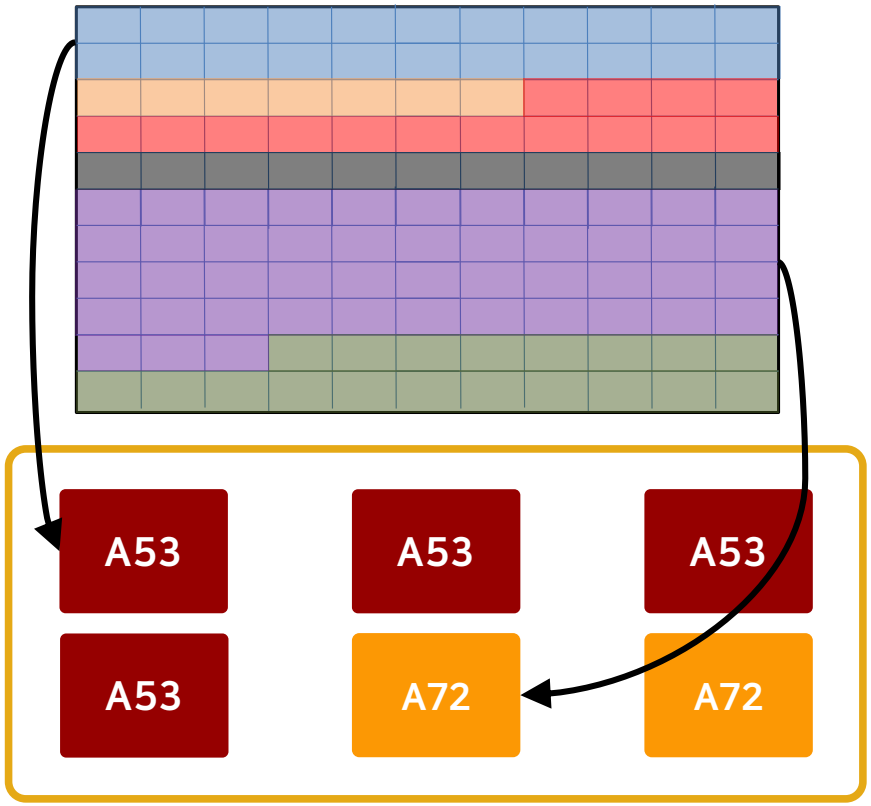
$$core = \{count(little), count(big), count(big.little)\}$$

$$w = (C_{max} - C_{min}) / C_{step}$$

$$_n H_r = Combinations\_with\_repetition$$

$$totalcase = \sum_{k=1}^{len(kernel_{list})} \sum_{n=1}^{len(core)} core_n H_w$$

전체 작업량과 코어 별 분배 한 결과
'little+ big' 의 프로파일 한가지 예



05 결과

1. AlexNet

- ✓ 추론 성능 :
  - ✓ 기본 ArmCL에 대비 최대 35% 성능 향상
- ✓ 탐색 시간 :
  - ✓ 코어 별 최적 작업량 탐색 : 약 4시간
  - ✓ 전체 탐색 : 약 105 시간

AlexNet 최적 스케줄 탐색 시간	
추론 시간 (단위 : ms)	탐색 시간 (단위 : 초)
기본 ArmCL	0
코어 별 작업량 최적화	16,170
코어와 컨볼루션 최적화	378,666

레이어(Kernel)		기본 ArmCL (ms)	프로파일 기반 최적화(ms)	성능 향상(%)
01	Convolution	15.164	9.480	53.541
02	Mul	0.920	0.760	17.391
03	Normalize	4.640	2.803	39.591
04	Pool2d	0.527	0.506	3.985
05	Convolution	13.997	7.936	60.877
06	Convolution	13.854	7.841	61.188
07	Concatenate	0.465	0.409	12.043
08	Concatenate	0.469	0.412	12.154
09	Mul	0.585	0.496	15.214
10	Normalize	3.077	1.816	40.981
11	Pool2d	0.474	0.403	14.979
12	Convolution	8.306	6.002	27.739
13	Convolution	3.909	2.817	36.838
14	Convolution	3.885	2.859	35.727
15	Concatenate	0.269	0.197	26.766
16	Concatenate	0.283	0.199	29.682
17	Convolution	3.254	2.377	44.530
18	Convolution	3.235	2.235	43.771
19	Concatenate	0.233	0.165	29.185
20	Concatenate	0.236	0.161	31.780
21	Pool2d	0.208	0.197	5.288
22	Reshape	0.403	0.399	0.993
23	FullyConnected	23.166	21.769	6.030
24	FullyConnected	11.142	9.925	10.923
25	FullyConnected	2.737	2.604	4.859
26	SoftMax	0.026	0.026	0.000
Total		115.521	84.853	34.389

05 결과

2. VGG 16

- ✓ 추론 성능 :
  - ✓ 기본 ArmCL에 대비 최대 39% 성능 향상
- ✓ 탐색 시간 :
  - ✓ 코어 별 최적 작업량 탐색 : 약 4시간
  - ✓ 전체 탐색 : 약 135 시간

VGG16 최적 스케줄 탐색 시간	
추론 시간 (단위 : ms)	탐색 시간 (단위 : 초)
기본 ArmCL	0
코어 별 작업량 최적화	12,483
코어와 컨볼루션 최적화	486,039

레이어(Kernel)		기본 ArmCL (ms)	프로파일 기반 최적화(ms)	성능 향상(%)
01	Convolution	19.671	8.132	58.660
02	Convolution	126.269	93.668	25.819
03	Pool2d	3.89	3.548	8.792
04	Convolution	53.261	32.688	38.627
05	Convolution	82.459	47.158	42.810
06	Pool2d	1.998	1.993	0.250
07	Convolution	35.753	19.925	44.270
08	Convolution	59.937	32.874	45.152
09	Convolution	60.225	32.791	45.553
10	Pool2d	1.061	0.957	9.802
11	Convolution	37.561	17.137	54.376
12	Convolution	66.332	31.502	52.509
13	Convolution	65.523	31.364	52.133
14	Pool2d	0.626	0.372	40.575
15	Convolution	19.445	11.424	41.250
16	Convolution	19.373	11.438	40.959
17	Convolution	19.083	11.53	39.580
18	Pool2d	0.237	0.055	76.793
19	Reshape	0.666	0.515	22.673
20	FullyConnected	62.577	58.838	5.975
21	FullyConnected	11.3366	9.914	12.549
22	FullyConnected	2.729	2.476	9.271
23	SoftMax	0.026	0.026	0.000
Total		750.08	460.326	38.630

06 결론

이기종 멀티 코어 구조에 적합한 스케줄 탐색

성능 향상

AlexNet과 VGG16에서
최대 39% 성능 향상하였다.

Q. 추후 연구 방향

- 일부 코어만 활용하는 기존 ArmCL에서 모든 코어를 활용한다면 성능이 향상이 되는 것을 알 수 있다.

- 근래에 쿼드 클러스터 구조(X3, A715, A710, A510) 구현된 CPU가 출시되고 있다. 그래서 다양한 보드와 인공지능 모델의 테스트가 필요하다.

- 모든 컨볼루션 연산을 탐색시 100시간이 넘는 튜닝 과정이 필요하므로 이를 해결할 추가 연구가 필요하다.

준비한 발표는 여기까지 입니다.

질문이 있다면 편하게 해주십시오.

감사합니다.

동의대학교 차주형