AsyMo: Scalable and Efficient Deep-Learning Inference on Asymmetric Mobile CPUs

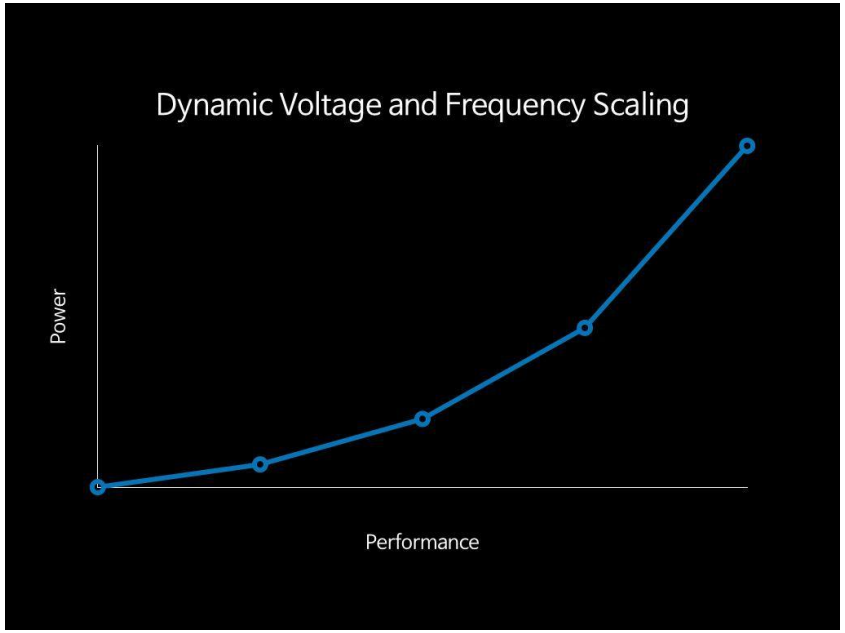
박준형 (dkdkernel@gmail.com)

배경

- MOBICOM 2021
- 주제 : CPU에서 DL 추론 성능에 향상에 대한 시스템 측면 내용

DVFS : Dynamic Voltage Frequency Scaling

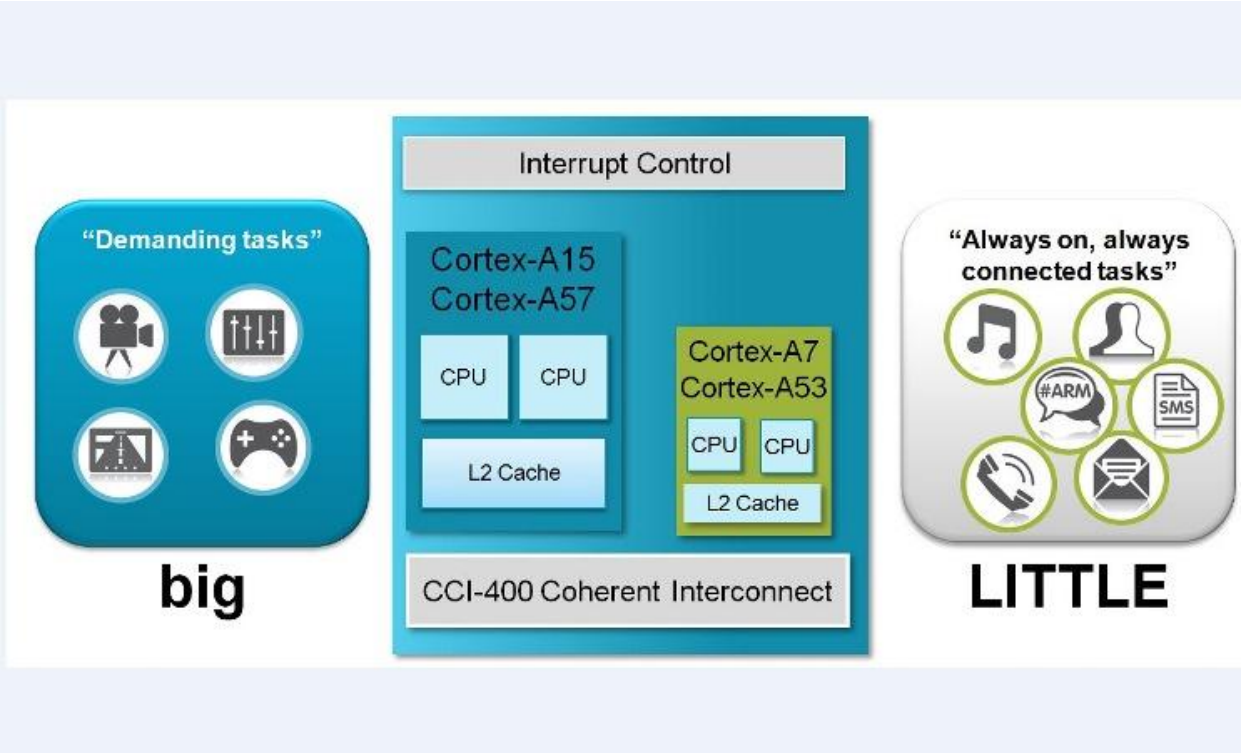
- 전력 절감 기술
- CPU 부하에 따라 Voltage와 Frequency를 조절
- Linux governo가 Policy 의해서 컨트롤됨



big.LITTLE & AMP

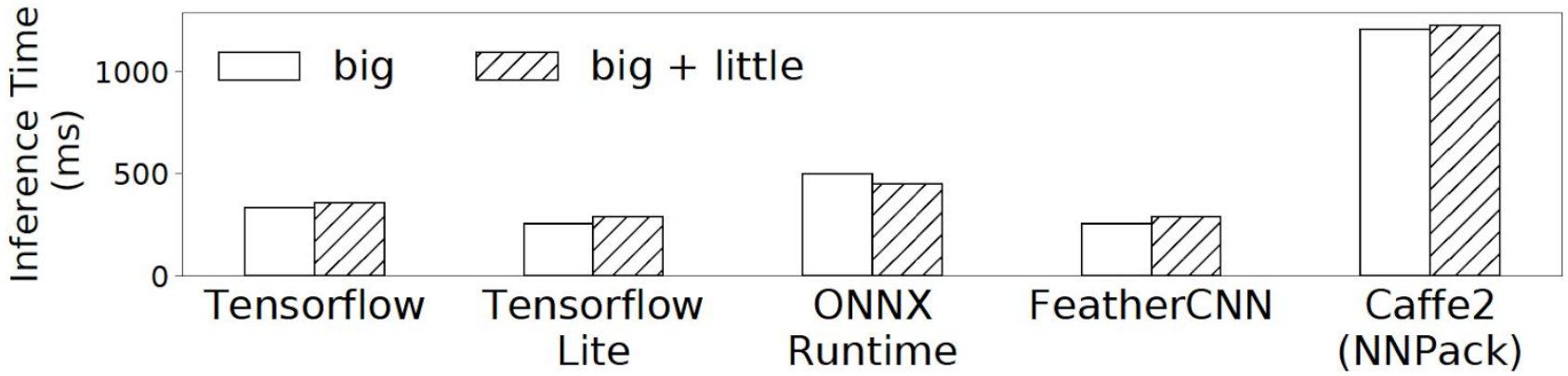
- **big.LITTLE**
- 저성능/고성능 멀티 코어 디자인
- 부하에 따라 동적으로 사용
- 에너지 절약, 발열 제어

- **AMP**(Asymmetric MultiProcessing)



Introduction

- ARM의 big.LITTLE같은 AMP(asymmetric multiprocessor 비대칭 멀티프로세서)에서 성능 확장성이 좋지 않음
  - big core만 쓰는게 big+LITTLE쓰는 것보다 빠른 경우도 있음
  - 원래 성능은 1.73배, 실제 실행 결과 5.63배
  - little이 더 적게 실행됨, 잘 활용되지 못함



Background : Parallelism in DL inference

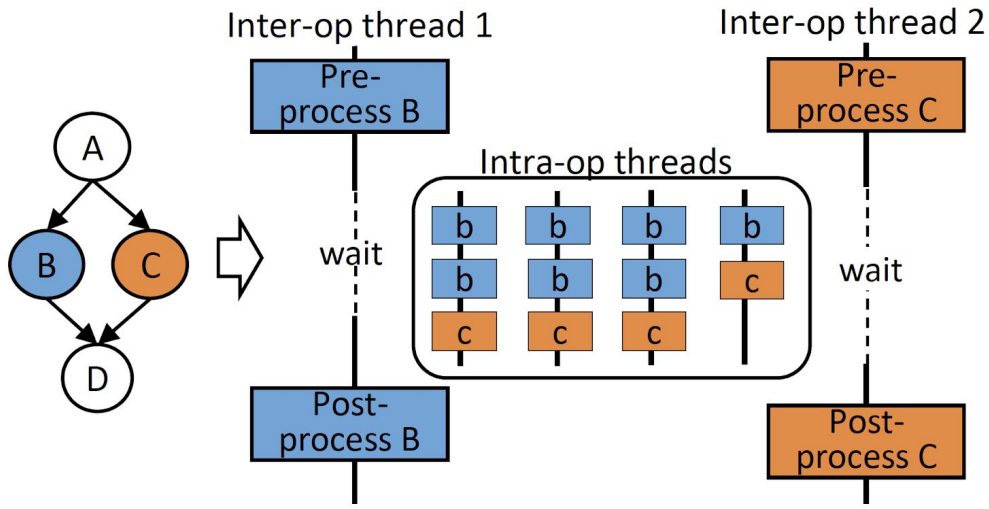


Figure 2: Inter- and intra-op parallel processing for op B and C in the example dataflow graph. The notation b and c show the partitioned tasks for B and C.

- B와 C같이 종속성 없는 op를 병렬로 처리
- 실행을 위해 공유 내부 작업 스레드 풀로 전송
- Eigen과 OpenMP같은 API는 AMP CPU를 고려하지 않고 각 스레드의 태스크 큐에 op의 태스크를 고르게 분배
- DL의 대부분은 MM, 쓰레드 분배가 AMP가 적절하지 않음

Background : MM partitioning

Table 1: The execution time % of MM in DL models

MobileNetsV1	SqueezeNet	ResNet-18	SSD-MobileNetV1	Char-RNN
65.76%	72.48%	83.84%	68.00%	97.45%
ResNet-50	ResNet-101	VGG-16	RNN-classifier	AlexNet
84.81%	88.27%	96.00%	99.11%	94.82%

```
for(i=0; i<M; i+=mc)
  for(j=0; j<N; j+=nc)
    threadpool->schedule([](){
      for(x=i; x<mc; x+=mr)
        for(y=j; y<nc; y+=nr)
          mm([mr,K],[K,nr])})
```

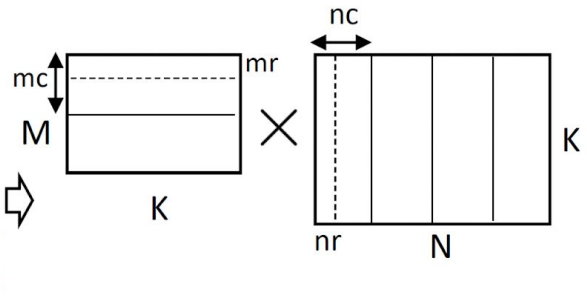


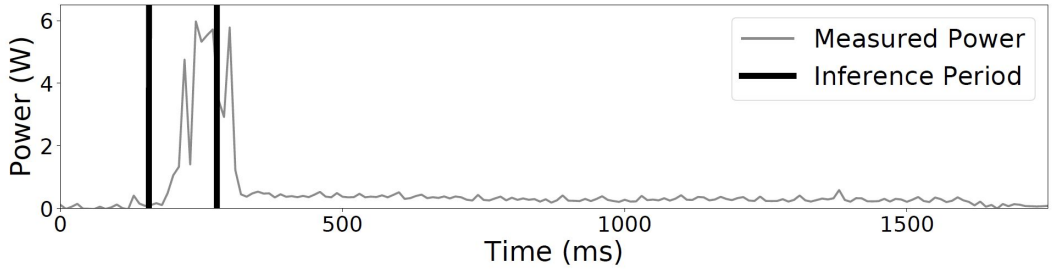
Figure 3: $(M,K) \times (K,N)$ is partitioned into $(mc,K) \times (K,nc)$ tasks for parallel execution. $(mr,K) \times (K,nr)$ is the basic computing block.

- MM partitioning
- MM은 DL에서 큰 비중을 차지하고 있음
- 기존의 partitioning은 같은 크기로 나누기 때문에 AMP에서는 효과적이지 않음

Background : Mobile AMP and OS DVFS

Mobile AMP and OS DVFS

- DVFS = Dynamic Voltage Frequency Scaling (동적 전압 스케일링)
- OS DVFS가 workload와 에너지 효율에 따라서 조절
- OS가 DL에 따른 CPU 클럭 설정을 제대로 못하고 있음
- 단기 추론에 대해 충분히 빠르게 반응하지 않음
- DL 연산과 DVFS의 클럭 설정 사이에 mismatch 있음
- 추론 시작 약 34ms이후에 점진적으로 증가함
- 추론 완료 25ms이후에 하강



AsyMo system design : System overview

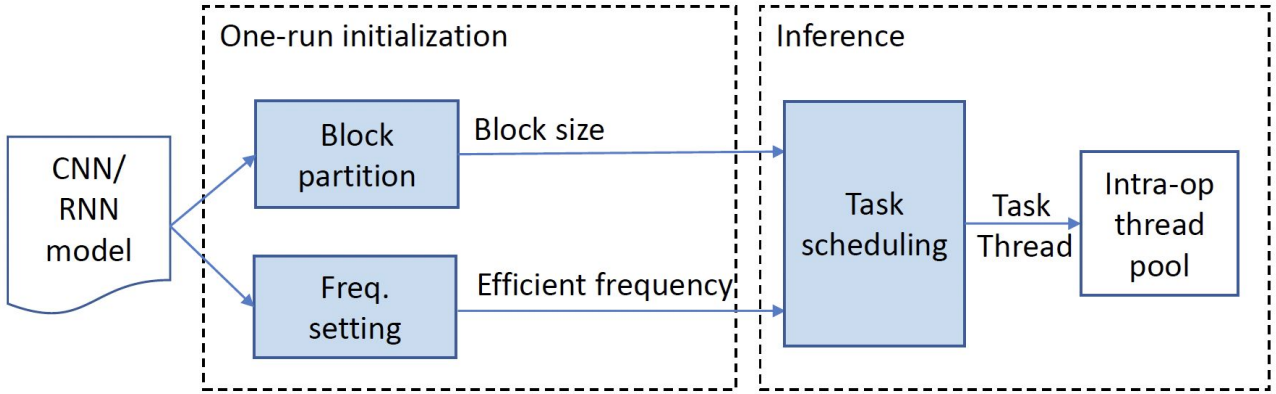


Figure 7: AsyMo (blue blocks) workflow.

- 초기화
  - 모델 로드 -> MM분할 -> 데이터 재사용/에너지-주파수 기반으로 최적 주파수 결정
- 추론 실행
  - 내부 작업 스레드를 하나의 CPU 코어에 바인딩하고 각 스레드에 공정하게 테스크 분배

1) latency-first 모드
2) energy-first 모드

AsyMo system design
: Cost-model-directed block partitioning

- **Design guidelines**
  - Task 크기
    - Bottleneck이 되지 않을만큼 작게 (너무 크면 Bottleneck)
    - Thrashing 나지 않을만큼 크게 (너무 작으면 I/O가 많아지고 Thrashing)
    - DL의 입력, 필터, 출력 크기는 특정 범위안에 있어 예측 가능
    - 추론 전에 오프라인 학습 한 번 해서 알아낼 수 있음

AsyMo system design
: Cost-model-directed block partitioning

- **Partitioning for big and little processors**
  - 행렬을 하위 행렬로 나눠서 프로세서 수준 분할 -> 코어 수준 분할
    - 이유1) 적정 크기가 프로세서마다 달라서
    - 이유2) 프로세서간 (불필요한 오버헤드인)데이터 전송을 방지 할 수 있어서
    - 분할 비율은 오프라인 실행으로 측정된 수치를 기반으로

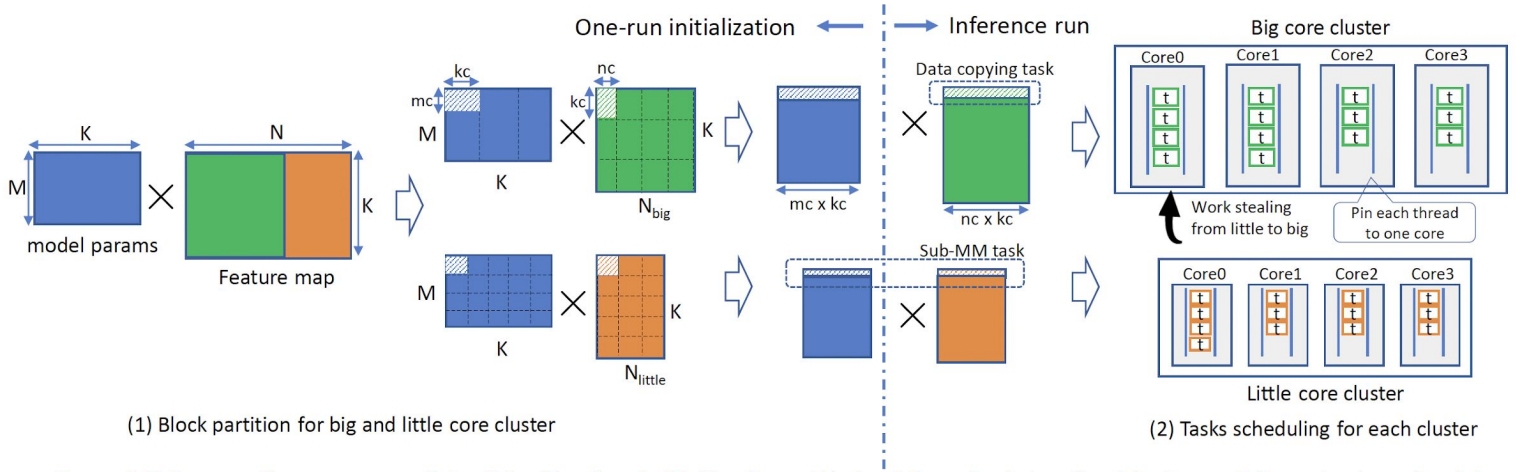
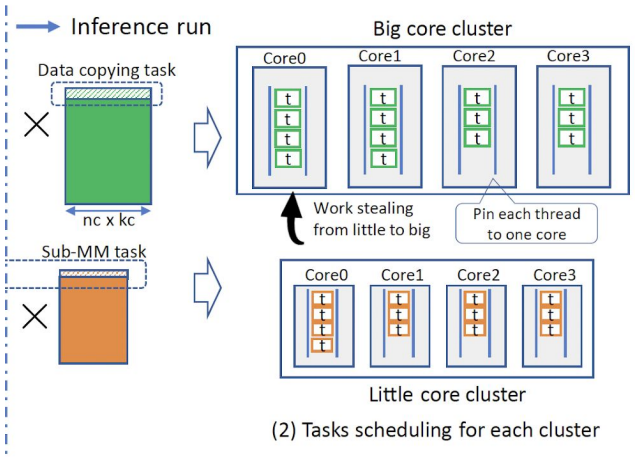


Figure 8: an MM execution process of AsyMo. During initialization, (1) AsyMo calculates the block partition strategy for big and little processor; (2) during inference, schedules the feature map copying and sub-MM tasks (notated by t in the figure) to each core within a processor.

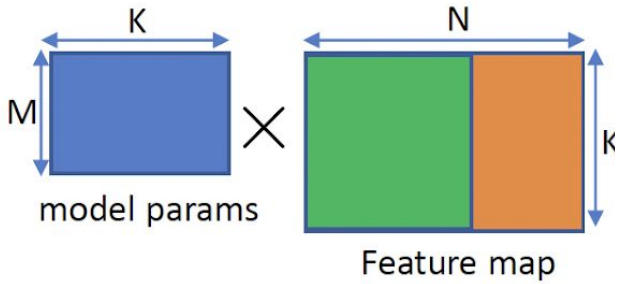
AsyMo system design
: Asymmetry-aware scheduling

- 스케쥴링 가이드
  - 코어와 프로세서마다 Task 수를 균형있게
  - 프로세서간 불필요한 데이터 이동 없도록
- AsyMo는 스레드 풀을 코어 번호와 동일하게 설정하고 OS 스레드 affinity를 사용하여 각 스레드를 코어에 바인딩함
- 가장 짧은 쓰레드 대기열에 예약
- 쓰레드 큐가 비어 있으면 Steal함



Frequency setting for energy efficiency : Design Considerations

- 데이터 재사용이 많을수록 필요한 메모리 액세스가 줄어듦
  - MV는 N 또는 M이 1 = 재사용성이 낮음 = 메모리-intensive
  - MM은 N 또는 M이 1보다 큼 = 재사용성 높음 = 컴퓨팅- intensive
- 컴퓨팅-intensive workload는 높은 CPU 주파수가 좋음
- 메모리-intensive workload는 낮은 CPU 주파수가 좋음
- Conv는 컴퓨팅 집중적, FC는 메모리 집중적 workload
- 오프라인 벤치마크로 주파수-에너지 곡선, 최소 에너지 주파수 찾기



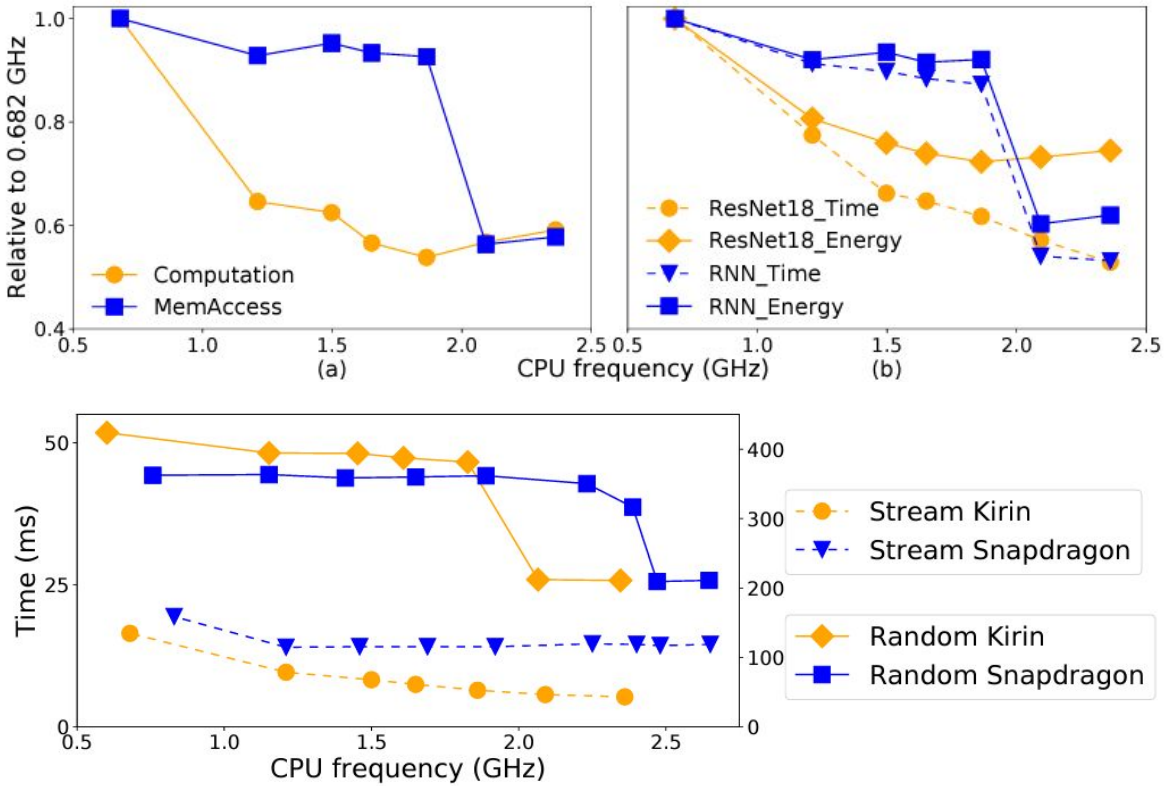
$$\mathrm{MM}\ (M,K)\times(K,N)$$

$$Data\_reuse = \frac{2MNK}{MK+NK+2MN} = \frac{2}{\frac{1}{N}+\frac{1}{M}+\frac{2}{K}}$$

Frequency setting for energy efficiency : Experimental verification

- 에너지 곡선에 따르면 RNN에서 주파수가 커지면 성능이 좋아짐
- 특정 CPU 주파수가 넘으면 랜덤 메모리 엑세스 시간 급감소

- RNN의 경우 1.86GHz 이후에는 큰 시간과 에너지 저하됨
- (아래) 랜덤 메모리 액세스 시간을 보면 1.86에서 2.09GHz 구간에서 대기 시간 빨라짐
- ARM CPU의 경우 특정 CPU 주파수에서 랜덤 메모리 액세스 대기 시간이 떨어집니다.
- 이게 메모리 집약적 RNN이 더 높은 주파수에서 훨씬 낮은 에너지 비용을 갖는 이유



Cost model training and energy profiling

- **Cost model training**
  - VGG-16, MobilNetV1에 대해서 input, filter, feature map 크기로 학습
  - 선형 회귀 모델로 학습, Kirin 970에서 1시간 정도 학습
- **Profiling of energy-frequency curves**
  - 안드로이드로 실험
  - LITTLE-core 주파수 낮추는 게 에너지에 도움되지 않음
  - 7%차이밖에 나지 않음

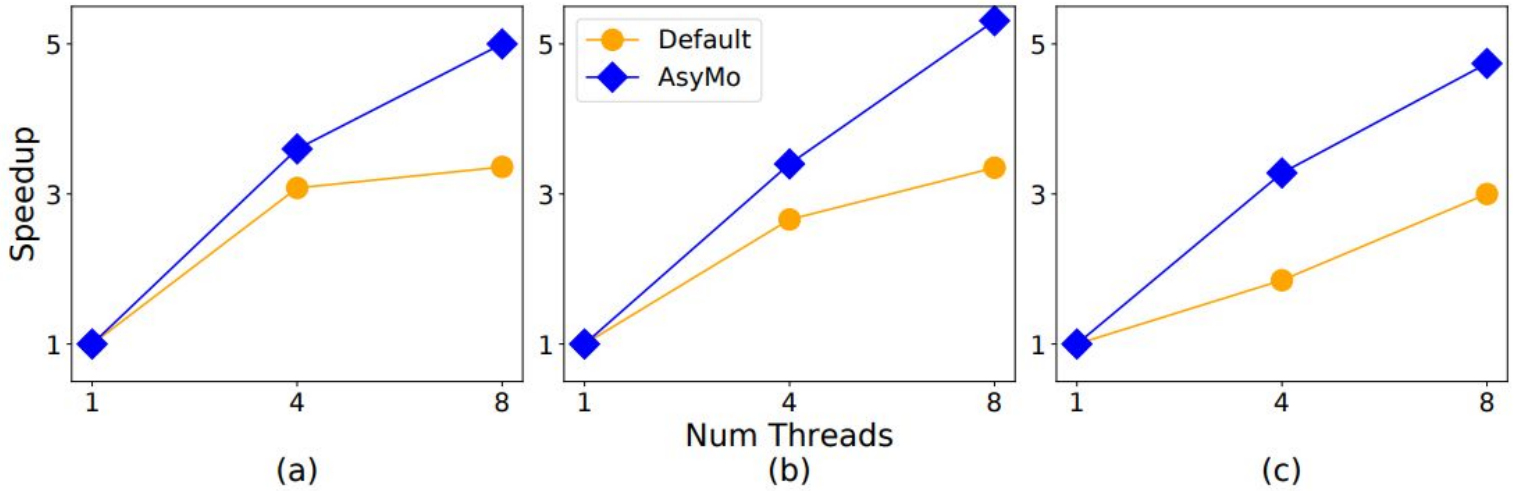
Evaluation : Experimental methodology

- Conv, FC로 나눠서 실험
- 1초 간격 20회 반복 시간과 에너지 관점에서 측정
- 첫 실행은 제외

Evaluation : Results

MM results

- 쓰레드에 따른 성능 향상 있었음
- a) TF, b) FeatherCNN, c) ORT (onnx runtime format)
- 1~4 big, 5~8 LITTLE

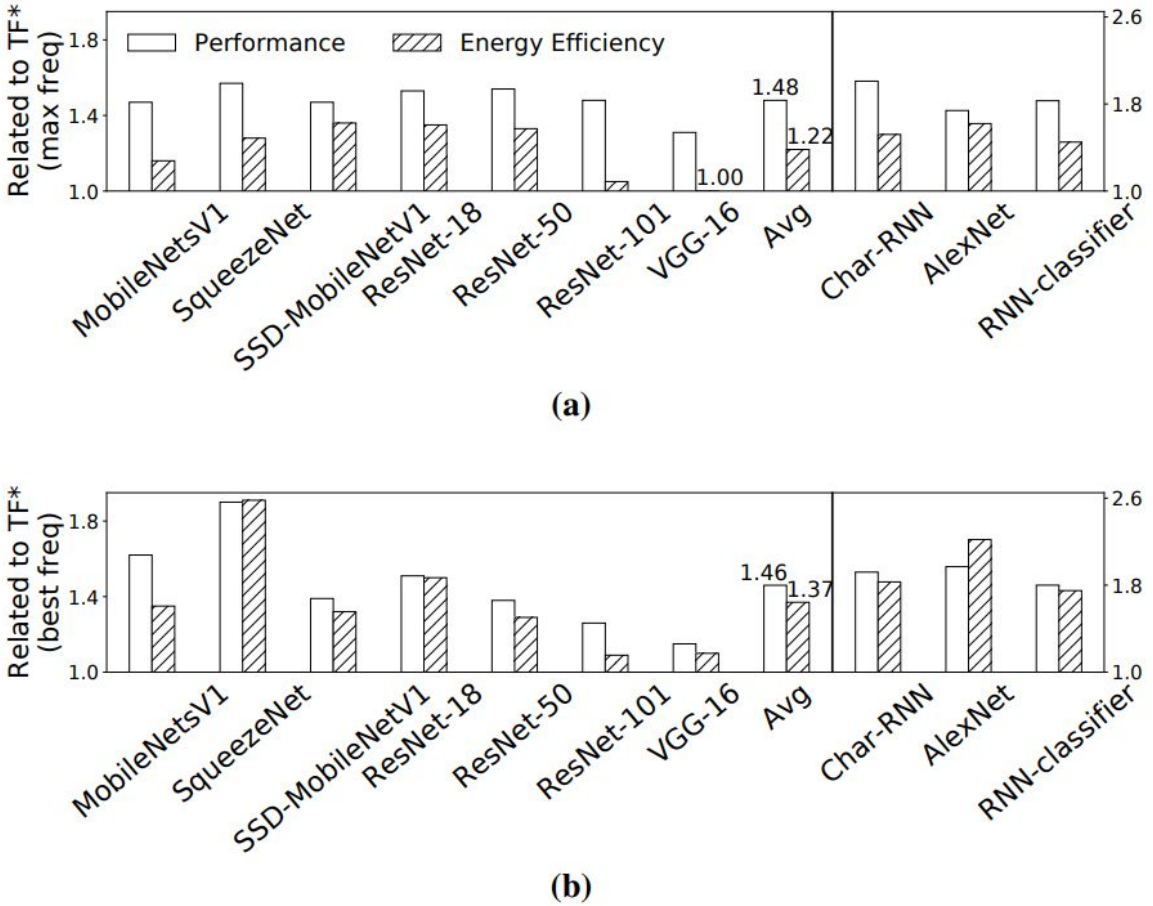


Evaluation : Results

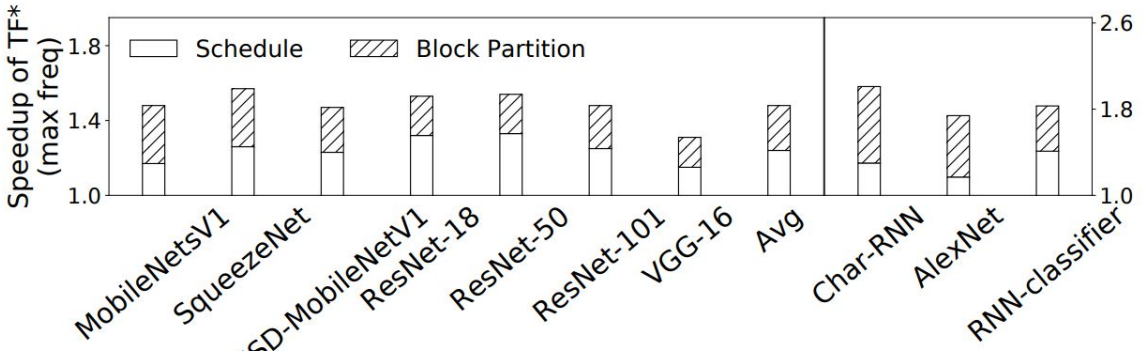
Model inference results

- TF대비 시간과 에너지 성능 향상
  - 최대 주파수보다 적게
  - Schedutil 연산 tail이 해결
  - 주파수 상승 지연 해결

- 좌측 Conv, 우측 FC
- a) 최대 주파수
- b) 최적 주파수



Evaluation : Results



Performance improvement breakdown

- **scheduling**
  - Conv-dominant에 대해서 24% 향상 (cache locality)
  - VGG-16은 LITTLE core 사용율이 이미 28%라 상승폭 적음
  - FC-dominant에 대해서는 8개로 나눈게 공정한 스케줄링을 확인하기 충분하지 않음
  - 블록 파티션 24% 향상
- **Block partition**
  - 평균 24% 향상
  - 작은 블록으로 균형 있는 병렬처리

Evaluation : Results

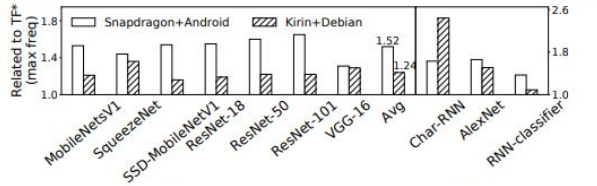


Figure 14: The performance improvement of AsyMo compared to TensorFlow⋆ on Snapdragon 845 with Android and Kirin 970 with Debian at max CPU frequency.

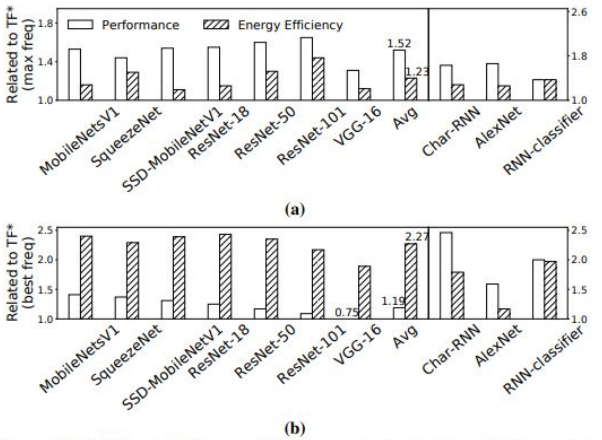


Figure 15: The relative performance and energy efficiency improvement of AsyMo on Snapdragon 845 with Android for Conv- (left axis) and FC-dominant (right axis) groups at (a) max CPU frequency; (b) most efficient frequency for AsyMo and *Schedutil* for TensorFlow⋆.

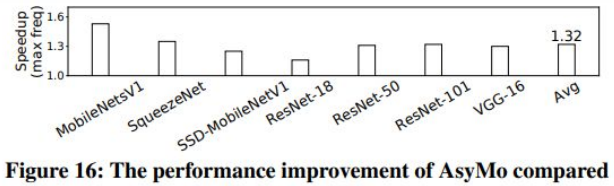


Figure 16: The performance improvement of AsyMo compared to TFLite⋆ on Kirin 970 with Android.

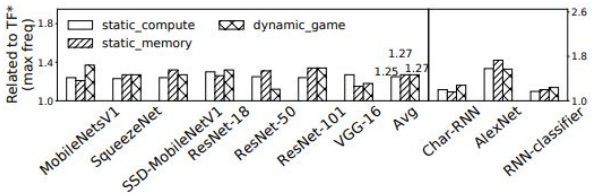


Figure 17: The relative performance improvement of AsyMo on TensorFlow⋆ with background load interference for Conv- (left axis) and FC-dominant (right axis) groups at max CPU frequency on Snapdragon 845 with Android.

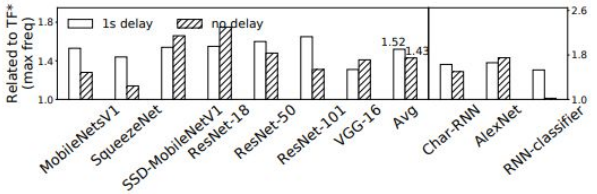


Figure 18: The relative performance improvement of AsyMo on TensorFlow⋆ w/o delay between inference runs for Conv- (left axis) and FC-dominant (right axis) groups on Snapdragon 845 with Android.

- Android & Debian
  - Android가 효과 더 큼
  - Debian은 이미 CPU 활용 잘 하고 있음
  - Debian은 Android보다 백그라운드 서비스가 적음
- 백그라운드 로드가 있는 경우
- 연속적으로 추론하는 경우

Conclusion

- 모바일 DL 추론을 위한 Task 분배 불균형;
  DVFS 불일치로 인한 에너지 비효율성 문제가 있었음
- MM 블록을 적절하게 분할, 공정한 Task 예약, 효율적 주파수 설정하는 AsyMo 제안
- 기존 DL 프레임워크 대비 성능과 에너지 효율성을 크게 향상

끝.

감사합니다.