Distributed Perception by Collaborative Robots

Interactive Talk

Ramyad Hadidi*, Jiashen Cao*, Matthew Woodward*, Michael S. Ryoo**, and Hyesoon Kim*

*Georgia Institute of Technology

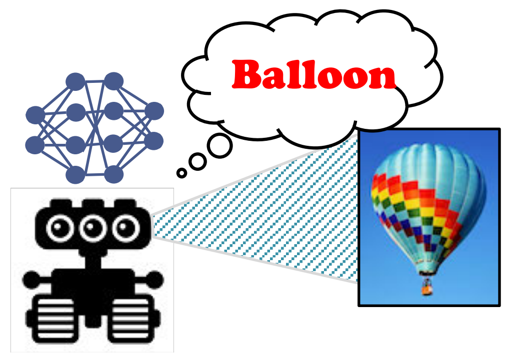
** EgoVid Inc.

Robots and Their Environment

- Robots need to process lots of raw data in their environment.
  - Visual, Sounds, Temperature, …
  - They need to understand it, to act upon it
  - For instance:
    - Drones that study an area after a disaster
    - Smart security system with lots of cameras
    - Swarm robots

Deep Learning (DL) and Robots

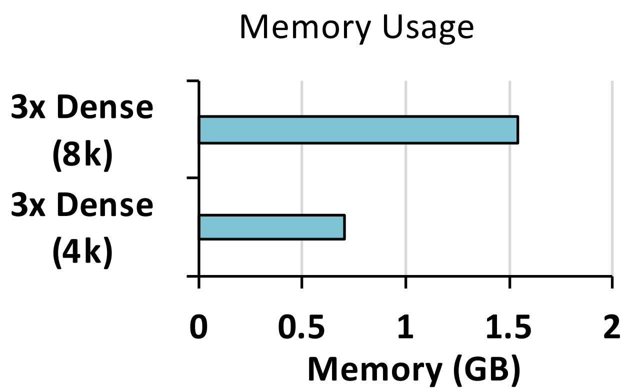
- How they should process complex raw data?
  - Use deep learning!
  - It can help in many tasks:
    - Object detection
    - Scene recognition
    - Action recognition
    - Speech recognition

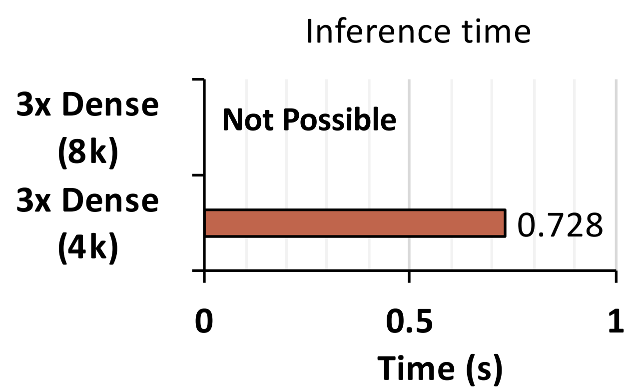


DL Computation is Heavy

- But DL models are computationally intensive and resource hungry specially for cheap robots.

An example of 3x dense layers on resource constrained device:





DL Computation is Heavy

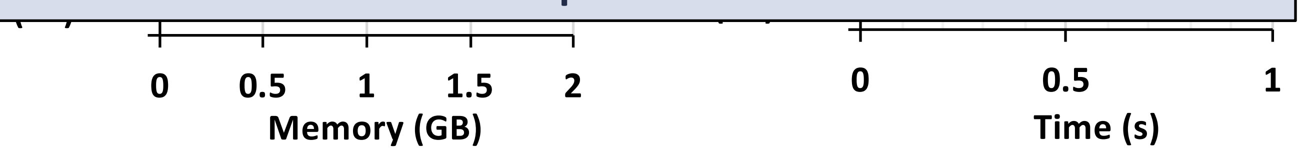
- But DL models are computationally
  inter bots.

An examp

It is difficult to execute DL models on all kinds of robots because:

1) Usually good models have large memory footprint.

2) For a robot, latency for single inference is important.

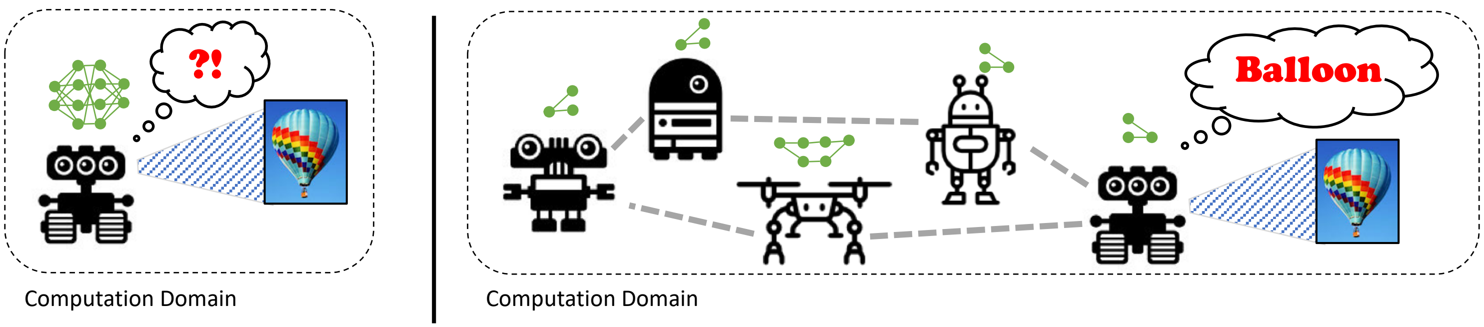


DL Computation is Heavy

- So robots need the result fast and in real time!
- Then how resource-constrained robots can use DL to understand their surroundings?

Let's Collaborate

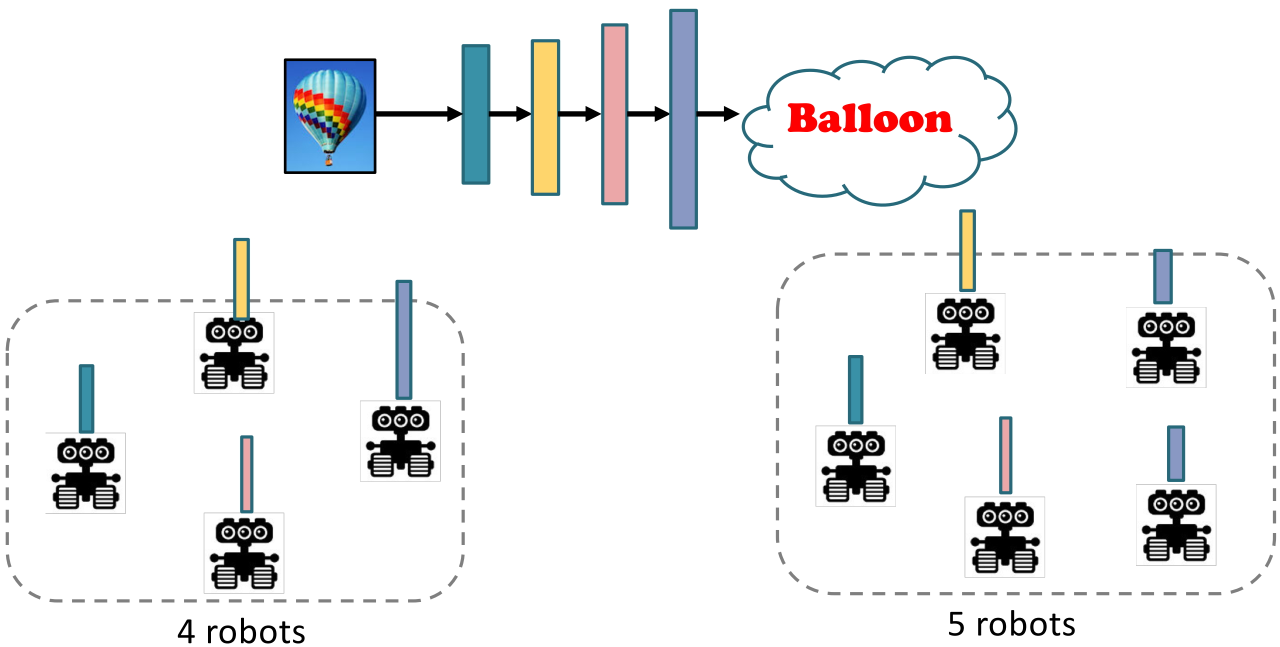
- Usually, many cheap robots share their environment.
- Not all robots need to perform computations at same time.
- So What if they share their knowledge and help each other?



(a) Single Robot **(b) Collaborative Robots**

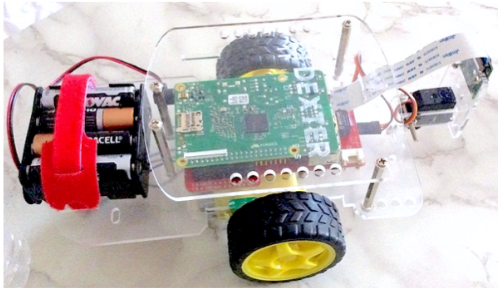
Our Work Overview

- We have proposed a technique to efficiently distribute DNN-based recognition.

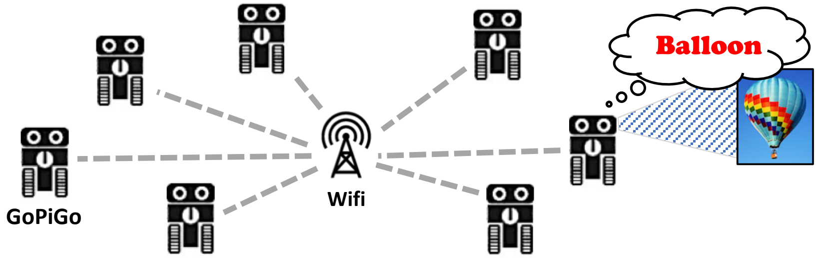


Our Work Overview

- We proposed an algorithm for deploying the distributed robot system only with Raspberry Pis.



(a) GoPiGo Robot



(b) Our Distributed Robot System

- We used AlexNet, VGG16, and a video recognition model as example models.

Outline

Introduction & Motivation

Data and Model Parallelism

- **Fully Connected and Conv Layers**

Distributing DNN

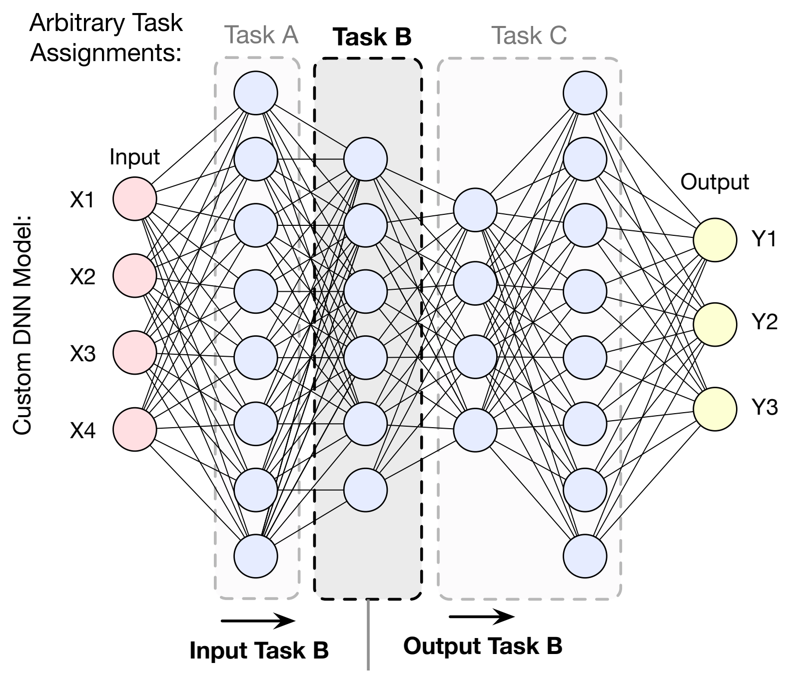
- Algorithm

System Evaluations

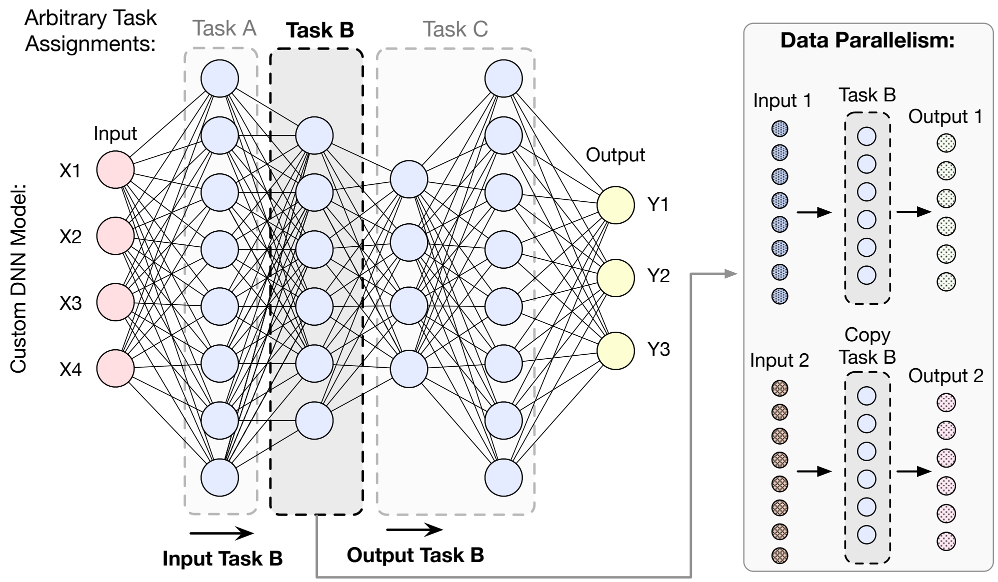
Conclusions

Model & Data Parallelism

Assume a custom DNN model, divided layer by layer:



Data Parallelism

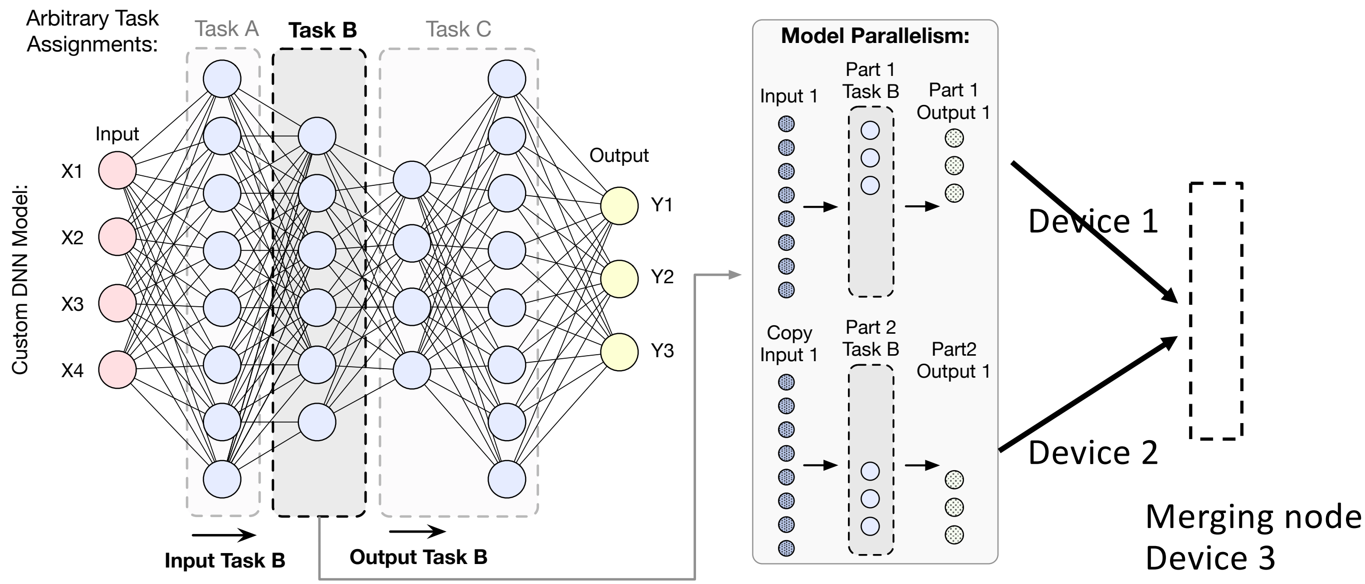


Device 1

Device 2

Data parallelism is providing the next input to multiple devices in a network.

Model Parallelism

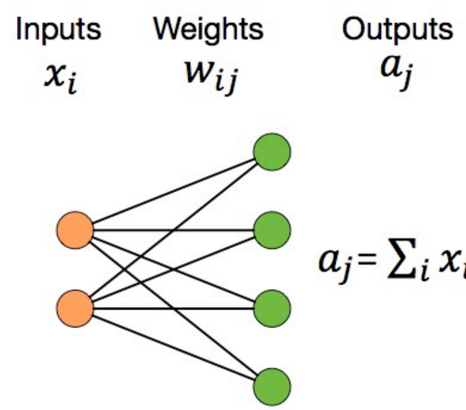


Model parallelism is splitting parts of a given layer or group of layers over multiple devices.

Fully Connected (FC) Layer

- Every output element is a summation of weighted inputs
- Each output element have its own set of weights
- A matrix multiplication

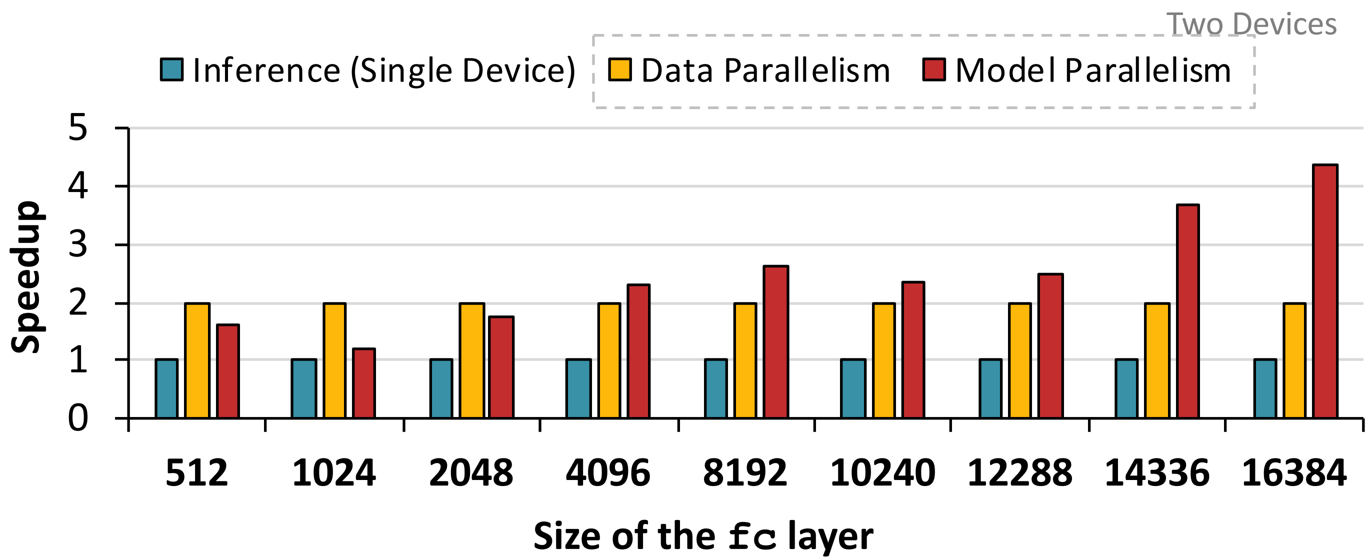
A simple 1-D example:



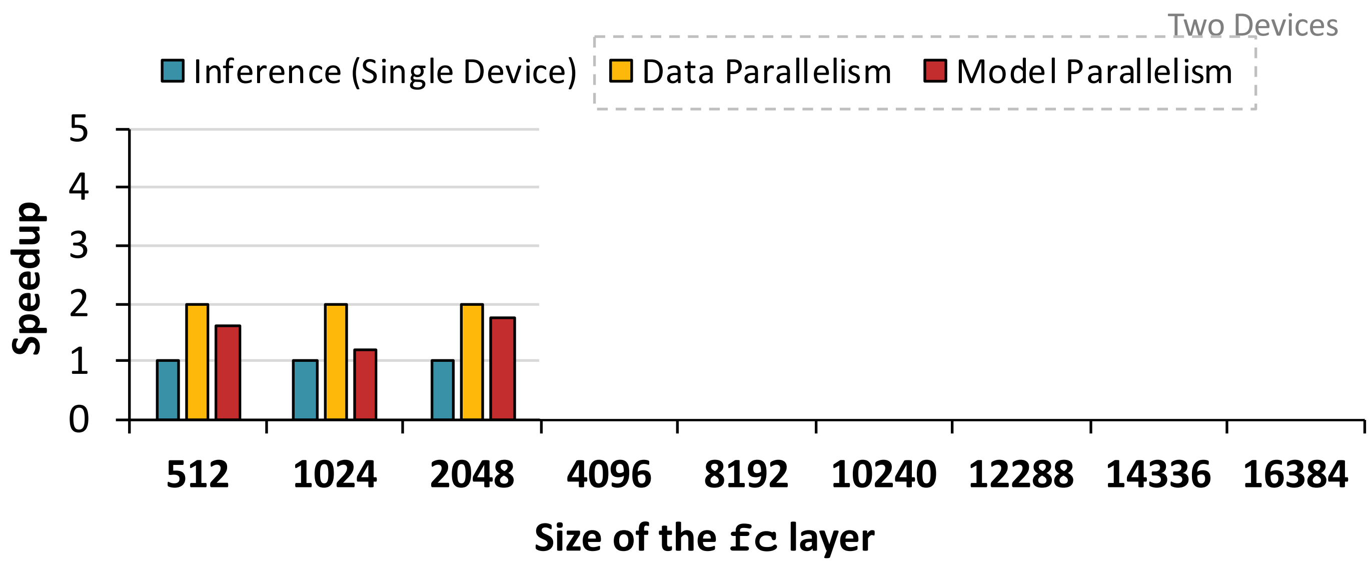
$$a_j = \sum_i x_i w_{ij}$$

Model & Data Parallelism for FC

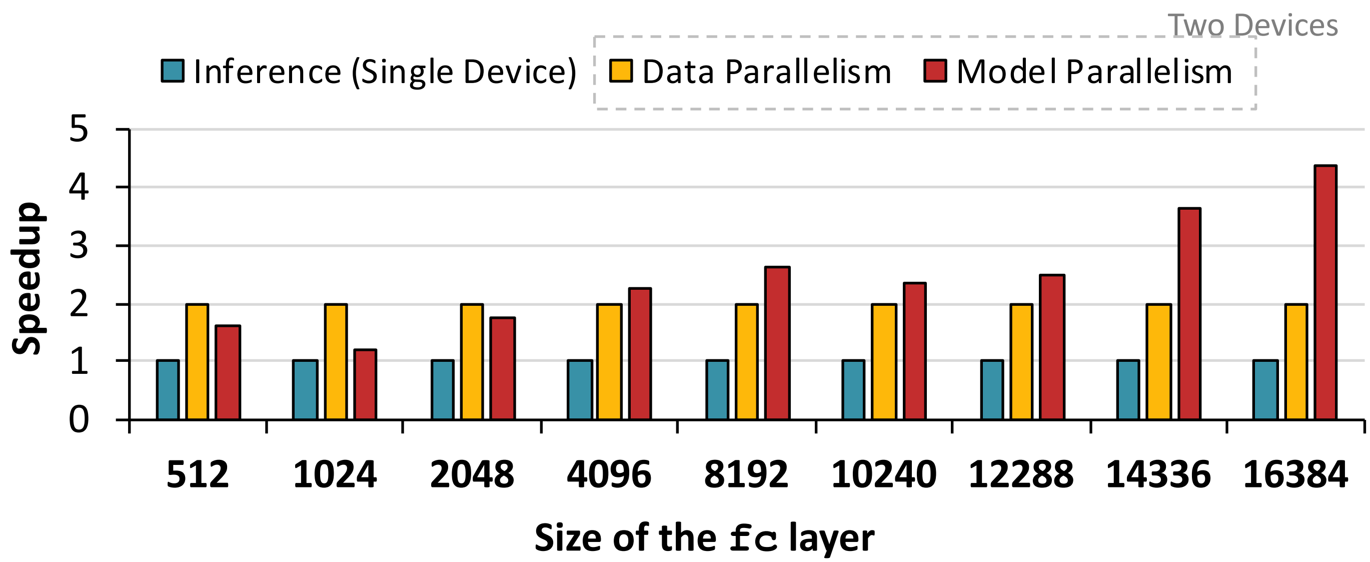
- Every output element is a summation of weighted inputs
- Each output element have its own set of weights
- A matrix multiplication



Model & Data Parallelism for FC

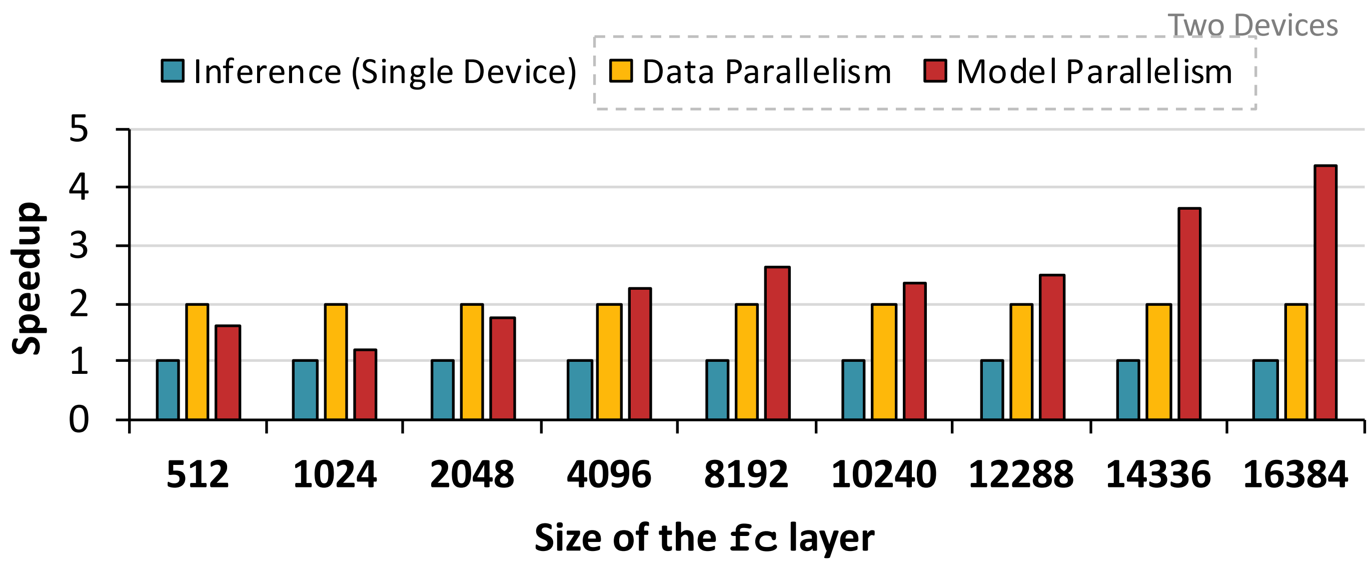


Model & Data Parallelism for FC



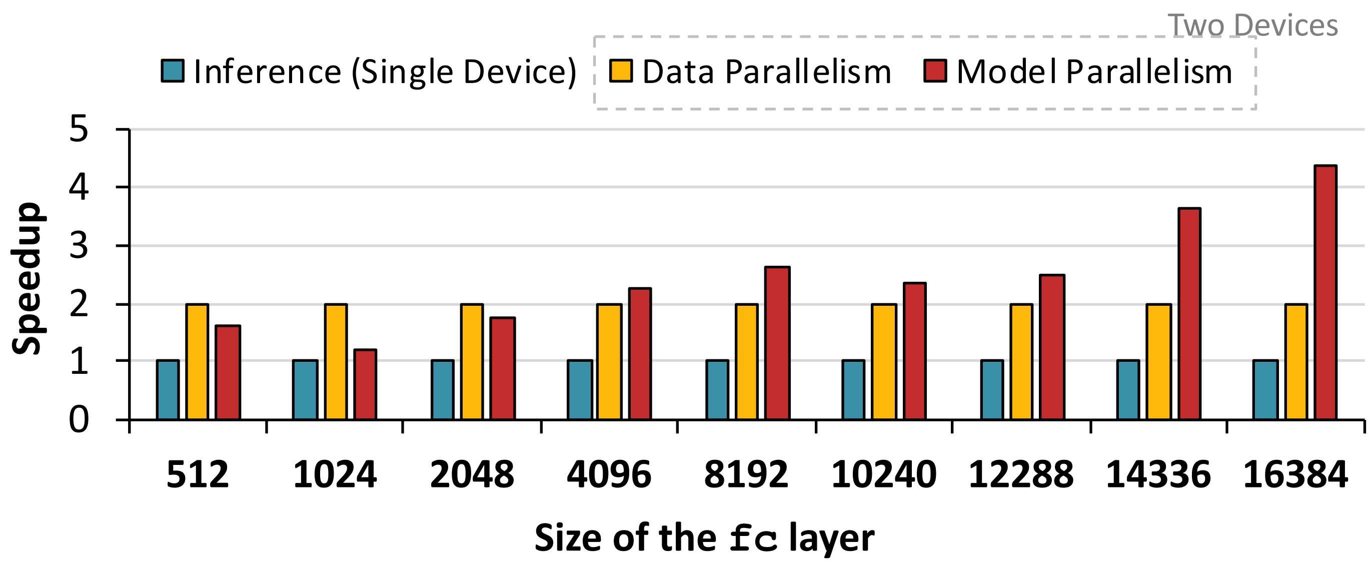
Georgia Tech comparch

Model & Data Parallelism for FC



Model parallelism reduces memory footprints.
So, we avoid slow hard drive accesses

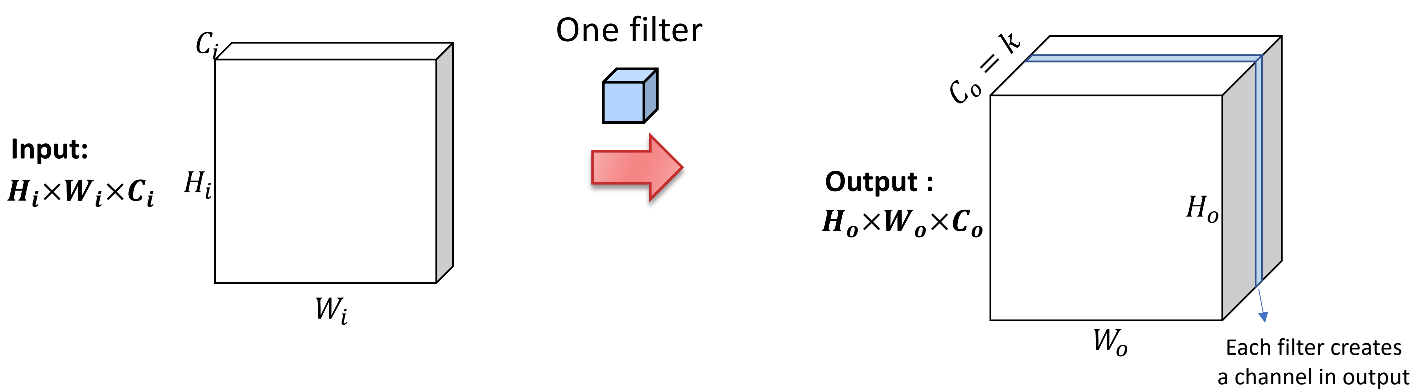
Model & Data Parallelism for FC



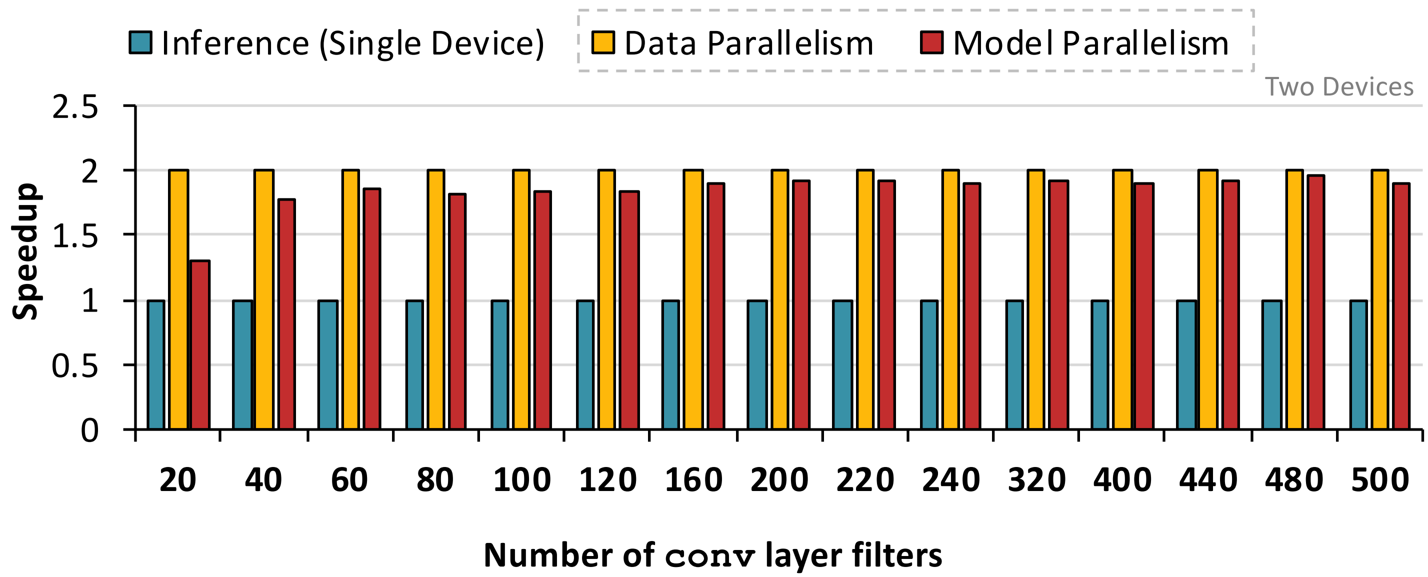
Fully Connected Layers: Either data or model parallelism depending on size of the layer, input, and memory

Convolution Layer

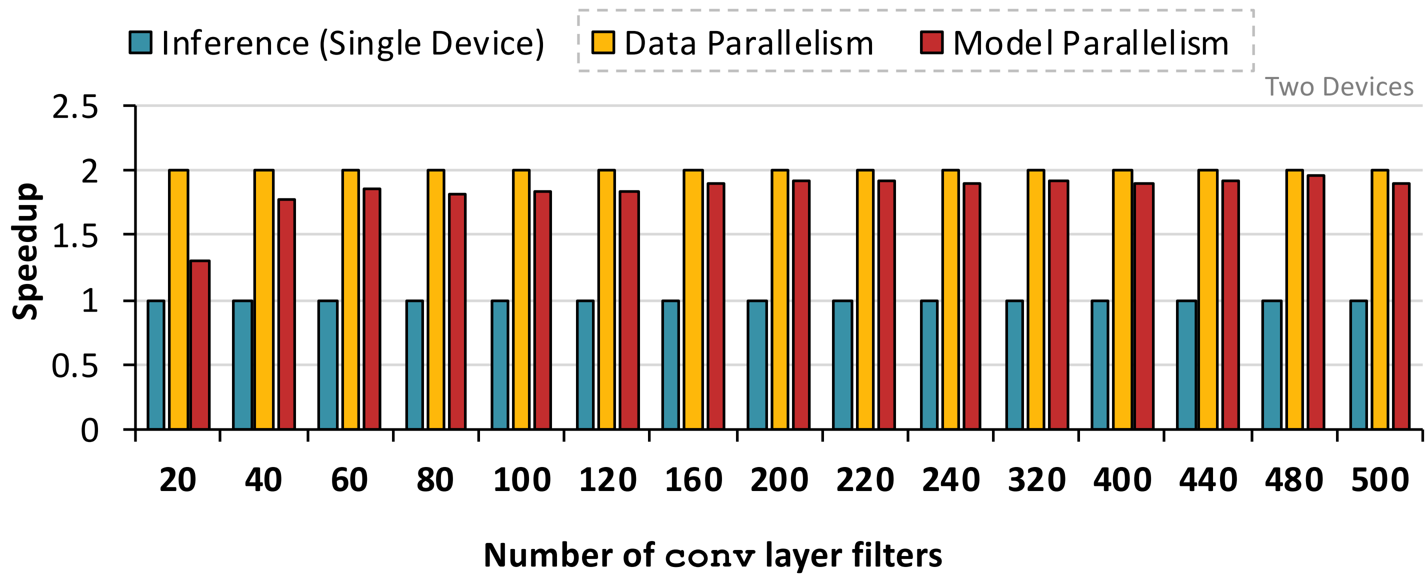
- Every channel in the output is derived from applying the same filter on the input
- Memory footprint size is smaller → fit into device memory size
- Model parallelism: split based on filters, and one more merging node at the end



Model & Data Parallelism for Conv



Model & Data Parallelism for Conv



Convolution Layers: Data parallelism is better

Outline

Motivation

Background

- ML Models Overview

Data and Model Parallelism

- Fully Connected and Conv Layers

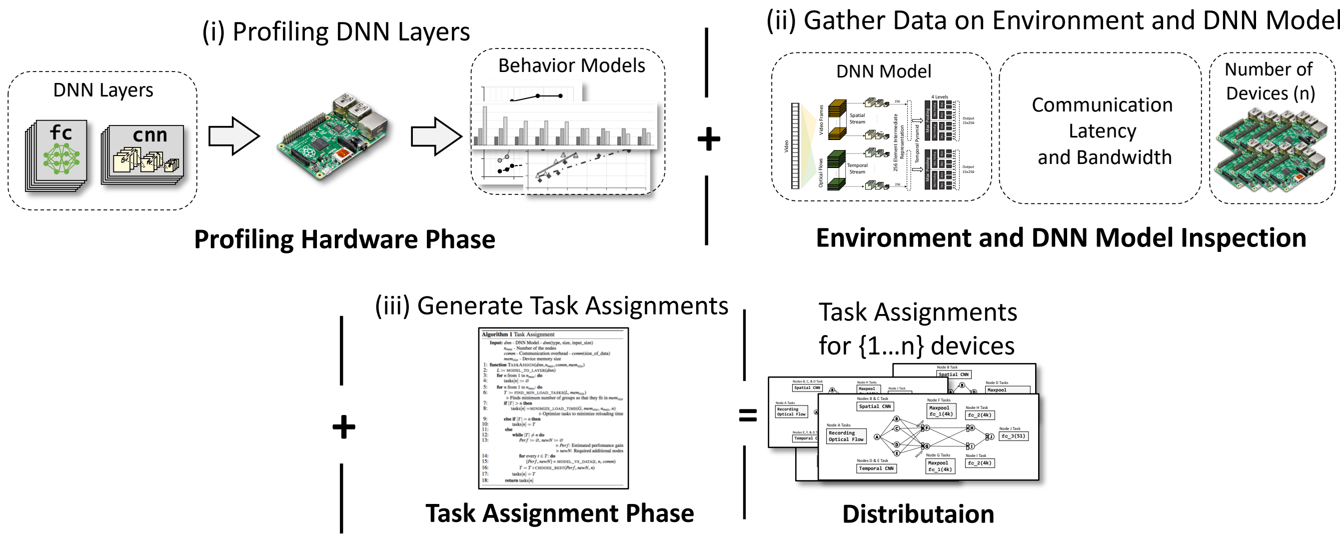
Distributing DNN

- **Algorithm overview**

System Evaluations

Conclusions

Distributing a DNN Model



Details are in the paper

Outline

- Motivation
- Background
  - ML Models Overview
- Data and Model Parallelism
  - Fully Connected and Conv Layers
- Distributing DNN
  - Algorithm overview
- **System Evaluations**
- Conclusions

Software & Hardware

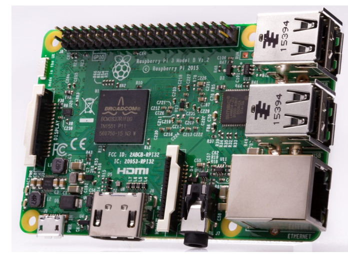
Software:

- Keras 2.1
  - With Tensorflow backend for Raspberry Pis
  - With Tensorflow-GPU backend for TX2
- Apache Avro for procedure call (RPC) and data serialization

Hardware Overview

Raspberry PI 3:

- Cheap and accessible platform
- Connected via a Wifi router
- No GPU
- $40



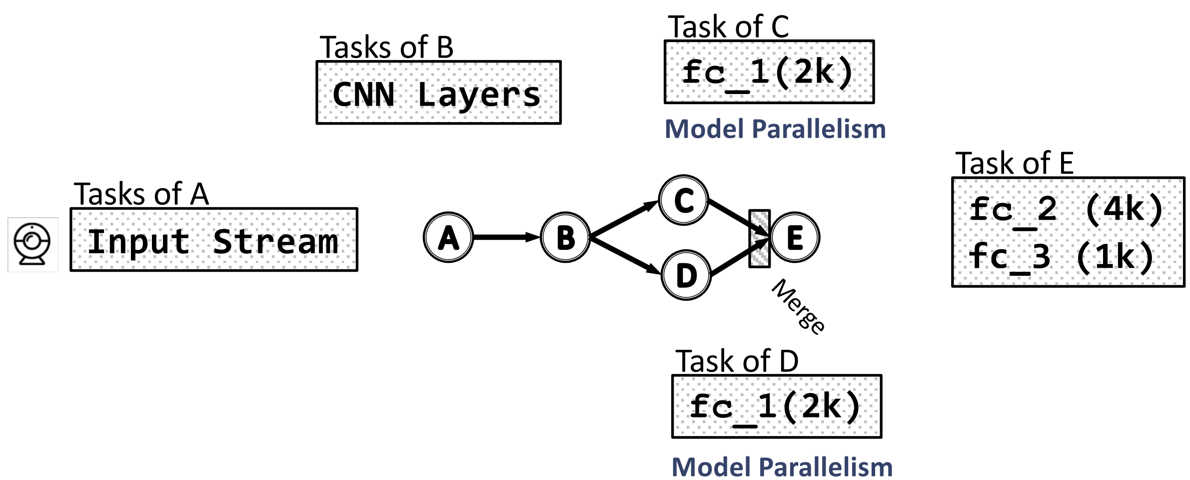
Nvidia Jetson TX2:

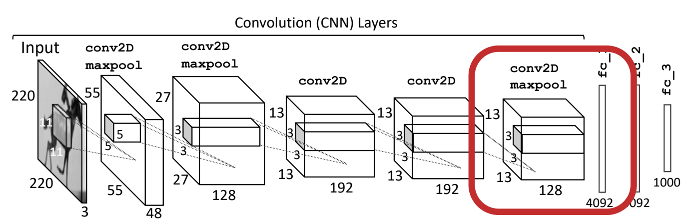
- High-end embedded platform
- Has a GPU
- $600

Moreover, we measured whole system power with a power analyzer

AlexNet Distribution I

Five-device system:





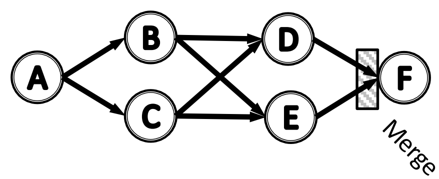
AlexNet Distribution II

Six-device system:

Tasks of B & C
`CNN Layers`
Data Parallelism

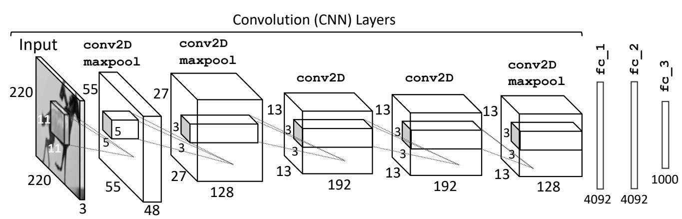
Task of D
`fc_1(2k)`
Model Parallelism

Tasks of A
`Input Stream`

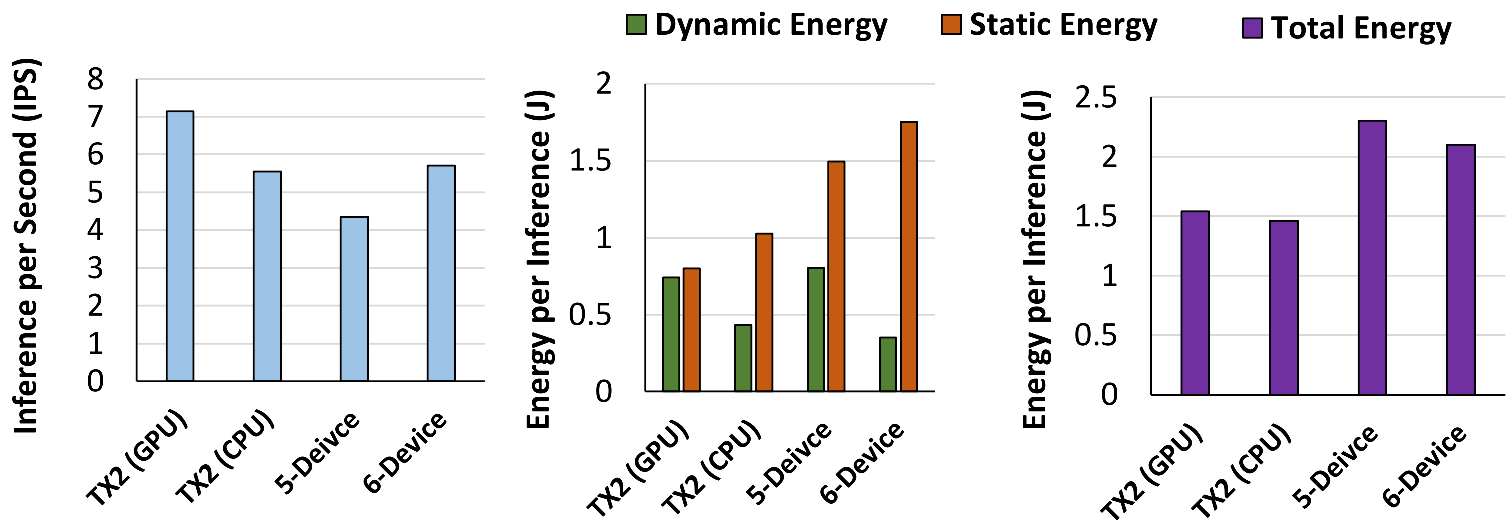


Task of F
`fc_2 (4k)`
`fc_3 (1k)`

Task of E
`fc_1(2k)`
Model Parallelism



AlexNet Results



IPS **Dynamic and Static Energy** **Total Energy**

Comparable IPS with TX2 (-30%)
Lower dynamic energy consumption

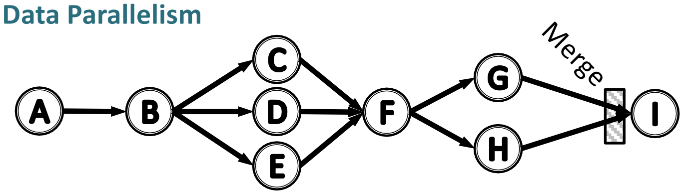
VGG16 Distribution I

Nine-device system:

Tasks of A
`Input Stream`

Tasks of C, D,& E
`Block 2,3,4`
Data Parallelism

Task of G
`fc_1(2k)`
Model Parallelism

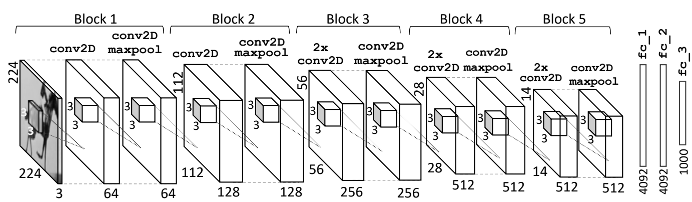


Task of I
`fc_2 (4K)`
`fc_3 (1k)`

Tasks of B
`Block 1`

Tasks of F
`Block 5`

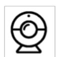
Task of H
`fc_1(2k)`
Model Parallelism



VGG16 Distribution II

11-device system:

Tasks of A

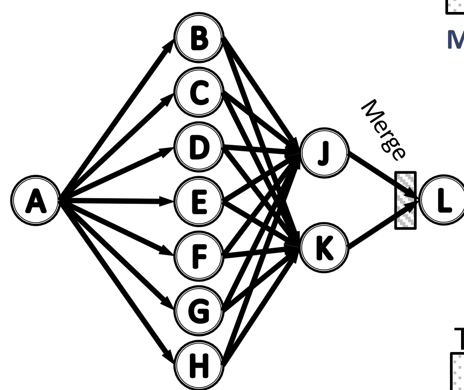


`Input Stream`

Tasks of B, C, D, E
F, G,& H

```
Block 1,2,
      3,4,5
```

Data Parallelism



Task of J

`fc_1(2k)`

Model Parallelism

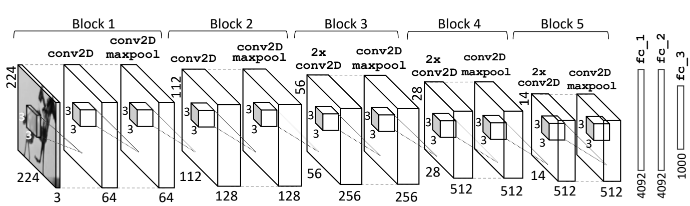
Task of L

```
fc_2 (4K)
fc_3 (1k)
```

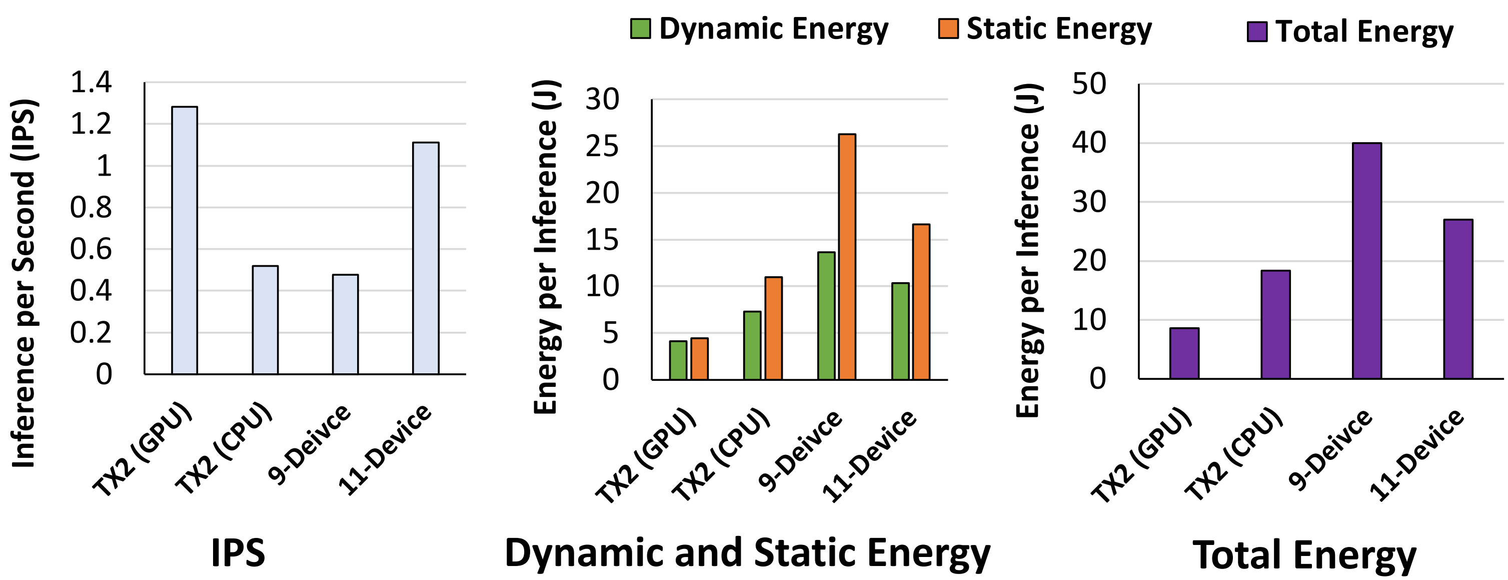
Task of K

`fc_1(2k)`

Model Parallelism



VGG16 Results



Comparable IPS with TX2 (-15%)

We achieve 2.3x speedup, by reassigning CNN blocks

Outline

Conclusions

- We used a farm of Raspberry PIs for DNN processing
- Our technique achieves acceptable real-time performance
  - Compared to TX2 CPU, we achieve similar performance with 6 robots for AlexNet
  - 11 robots for VGG-16 compared to TX2 GPU

Future Work:

- Study the robustness of such systems
- Apply our technique to more DNN models
- Implement our model on distributed robot systems

Backup Slides

Layers of ML Models

- Convolution: Applies several filters to the input
  - Compute bound, more locality
- Activation: Introduces non-linearity
  - e.g., ReLU $f(x) = \max(0, x)$, not compute intensive
- Fully Connected (Dense)
  - i.e., matrix multiplication, bandwidth bound
- Pooling
  - Reduces dimensions, simple doing max, average, and ... on a subset of input

Image Recognition Models (I)

- Single-stream AlexNet

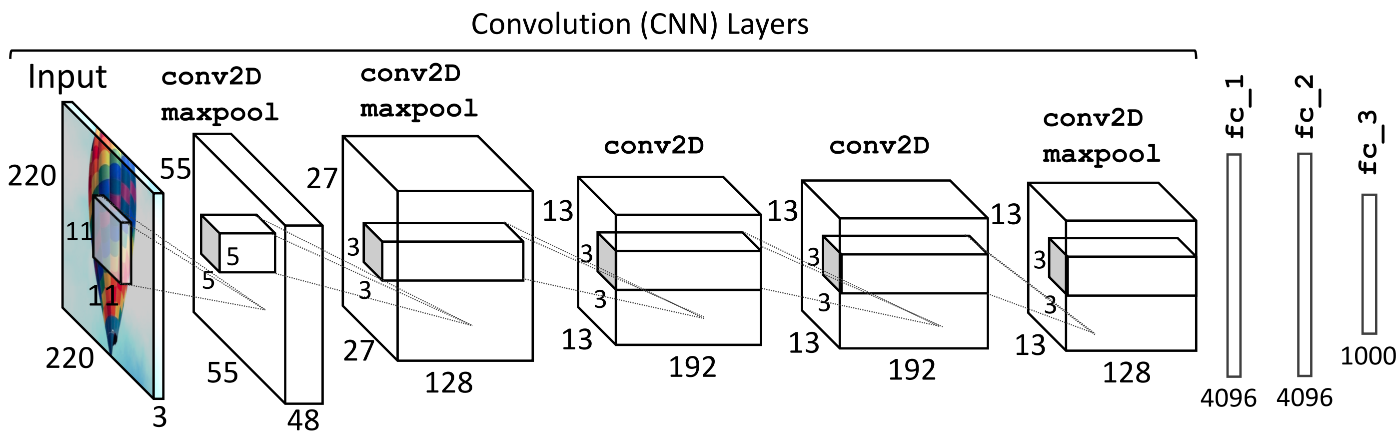
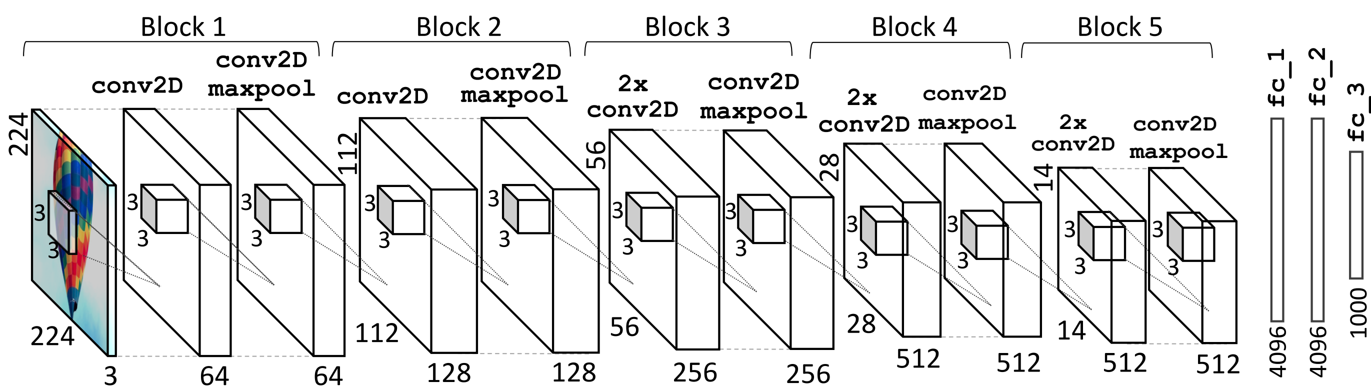


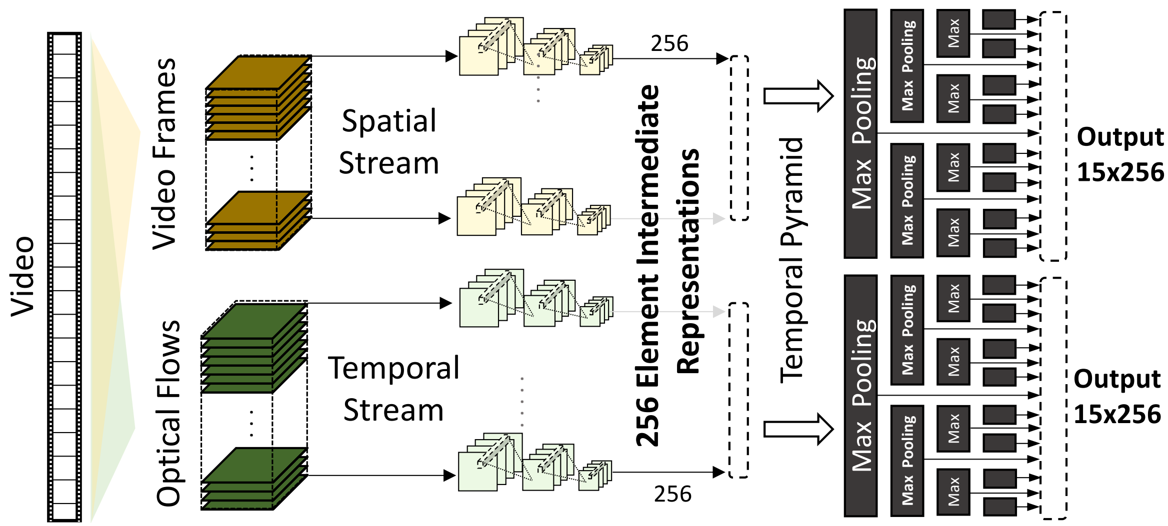
Image Recognition Models (II)

- VGG16



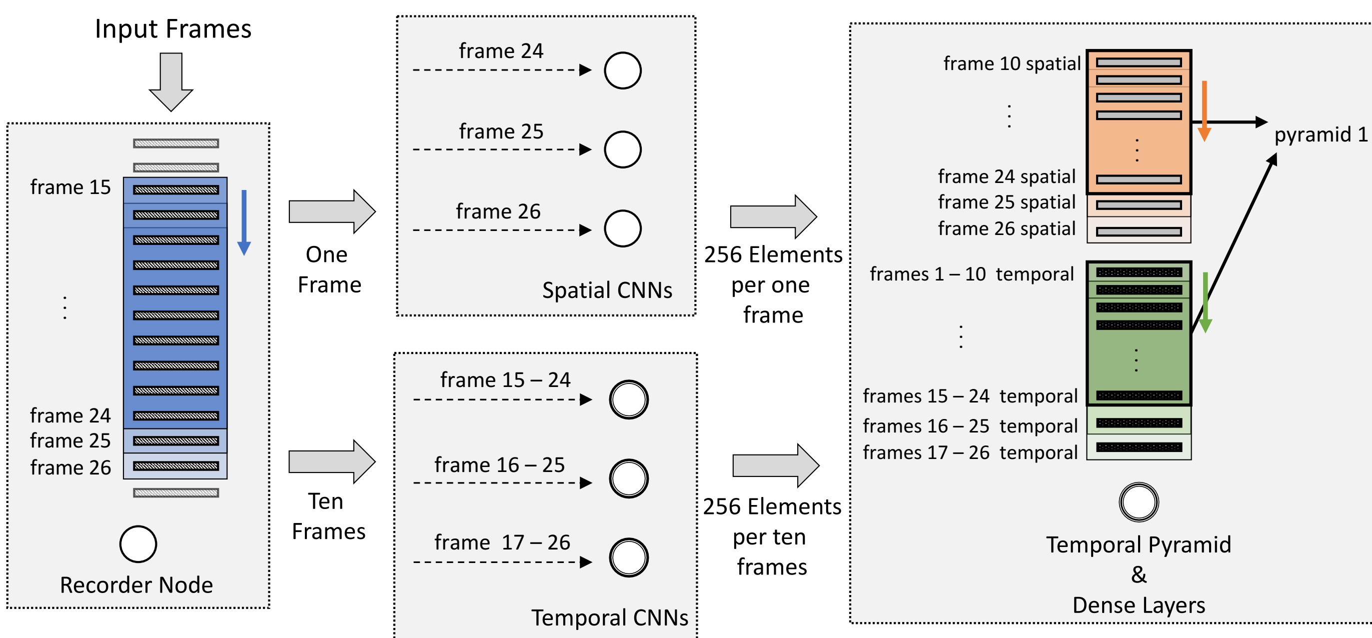
Vide Recognition Model

- i.e., Action recognition model
- Based on the two-stream model by Ryoo et al.[1]



M. S. Ryoo, K. Kim, and H. J. Yang, "Extreme Low Resolution Activity Recognition with Multi-Siamese Embedding Learning," in *AAAI'18*. IEEE, Feb. 2018.

Sliding Window



Algorithm

Algorithm 1 Task Assignment Algorithm.

function TASKASSIGNMENT($dnn, n_{max}, comm, mem_{size}$)
  Inputs: dnn: DNN model in form of layers[type, size, $input_{size}$, β]
    n_{max}: Maximum number of the devices
    $comm$: Communication overhead model ($comm(size_{data})$)
    mem_{size}: Device memory size
  $L :=$ EXTRACT_MODEL_TO_LAYERS(dnn)
  for n from 1 to n_{max}: **do**
    $tasks_{final}[n] := \varnothing$
  for n from 1 to n_{max}: **do**
    TG, $noFit$:= FIND_INITIAL_TASKGROUP(L, mem_{size})
    if $sizeof(TG) > n$ **then**
      tasks[n] =COMBINE_TASKS(TG, mem_{size}, n_{max}, n)
    if $sizeof(TG) = n$ **then**
      tasks[n] = TG
    if $sizeof(TG) < n$ or $noFit == True$ **then**
      while $sizeof(TG) \neq n$ **do**
        $task_{variant} := \varnothing$
        for every $t \in TG$: **do**
          [$task_{variant}$] += PROFILED_VARIANTS(t, $comm$)
        $task_{replaced}$, $task_{new}$ = SELECT_LOWEST([$task_{variant}$])
        $TG = TG - task_{replaced} + task_{new}$
      $tasks_{final}[n] = TG$
    return $tasks_{final}$

Georgia Tech comparch

Hardware Overview

Raspberry PI 3:

- Cheap and accessible platform
- Connected via a Wifi router
- No GPU
- $40

Table 1: Raspberry PI 3 specification

CPU	1.2 GHz Quad Core ARM Cortex-A53	
Memory	900 MHz 1 GB RAM LPDDR2	
GPU	No GPGPU Capability	
Price	$35 (Board) + $5 (SD Card)	
Power Consumption	Idle (No Power Gating)	1.3 W
	%100 Utilization	6.5 W
	Averaged Observed	3 W

Nvidia Jetson TX2:

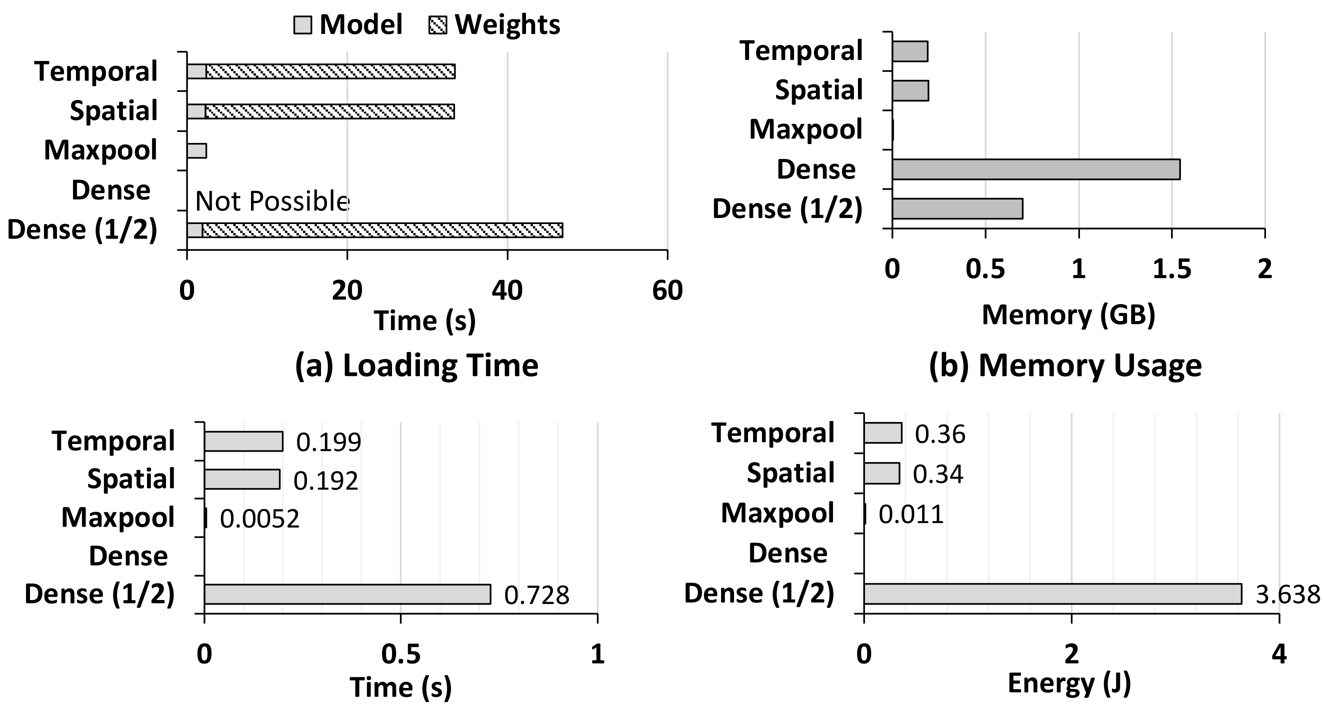
- High-end embedded platform
- Has a GPU
- $600

Table 2: Nvidia Jetson TX2 specifications

CPU	2.00 GHz Dual Denver 2 + 2.00 GHz Quad Core ARM Cortex-A57	
Memory	1600 MHz 8 GB RAM LPDDR4	
GPU	Pascal Architecture - 256 CUDA Core	
Total Price	$600	
Power Consumption	Idle (Power Gated)	5 W
	%100 Utilization	15 W
	Averaged Observed	9.5 W

Moreover, we measured whole system power with a power analyzer

Video Recognition on Single PI



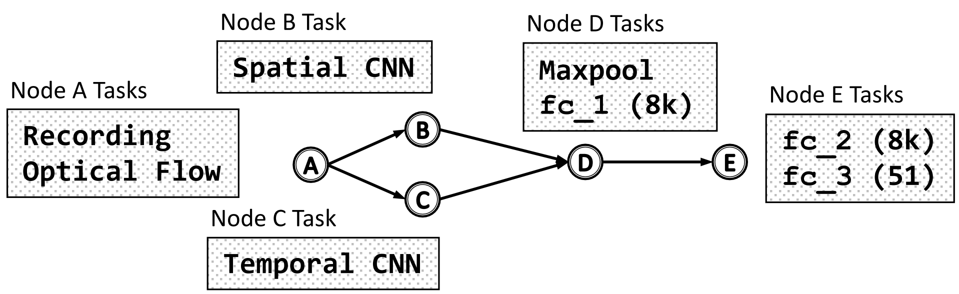
(a) Loading Time

(b) Memory Usage

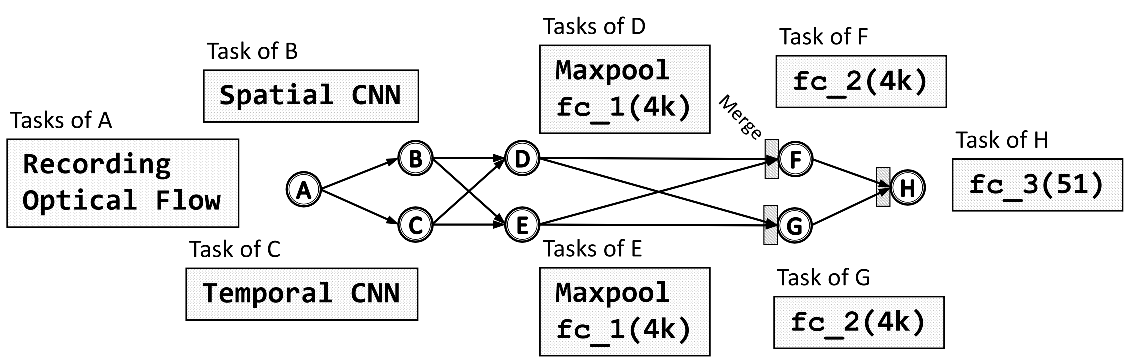
(c) Inference Time

(d) Energy Per Inference

Video Recognition Distributions (I)

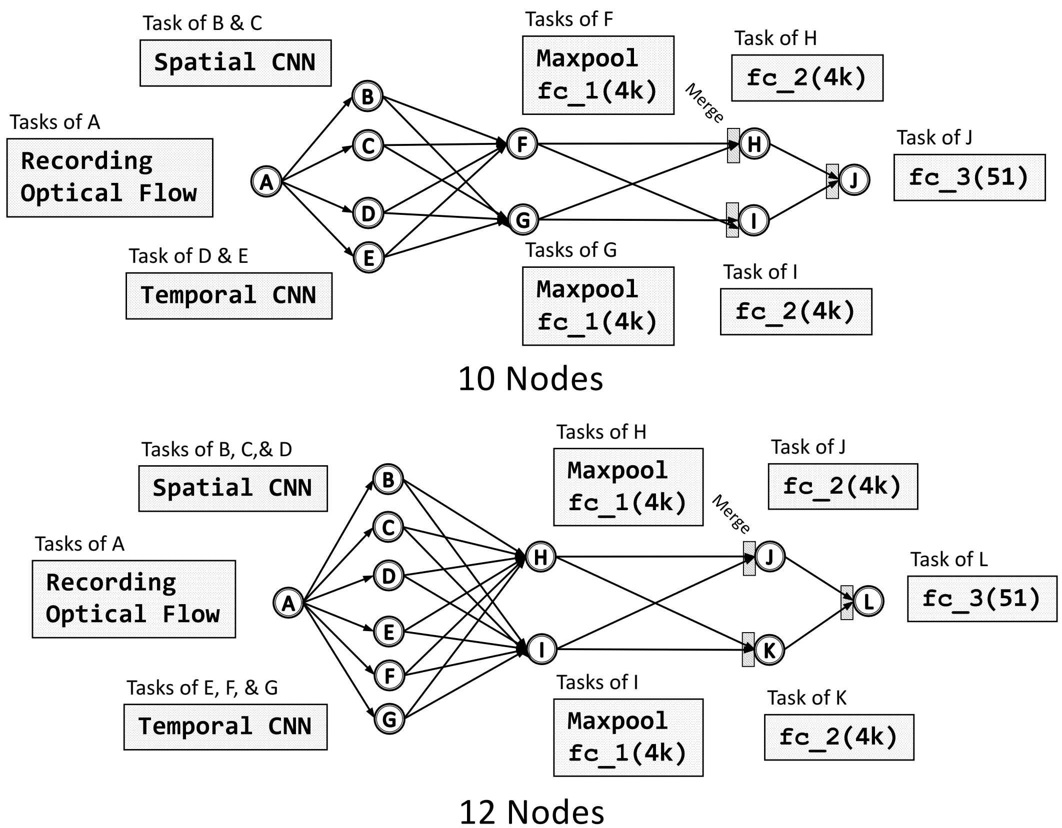


5 Nodes



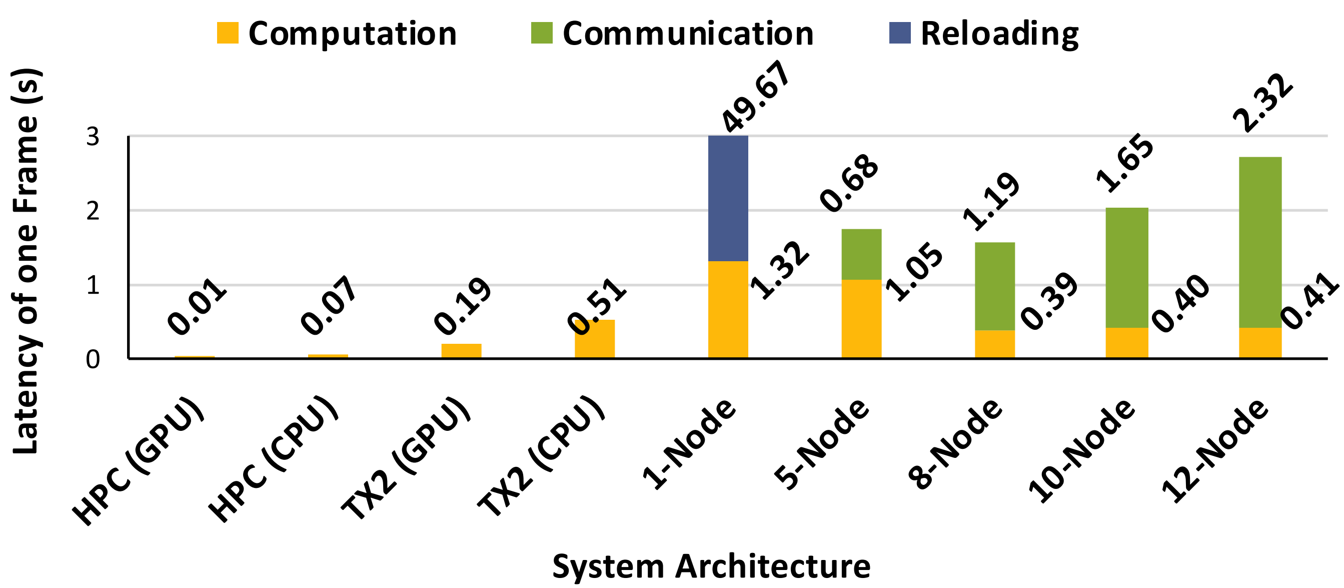
8 Nodes

Video Recognition Distribution (II)



Video Recognition Results (2)

Latency of one Frame (Seconds)



Video Recognition Results (3)

Energy: