Yüksek Başarımlı Hesaplama ve Yapay Zeka için Yazılım Çözümleri

Asst. Prof. Didem Unat

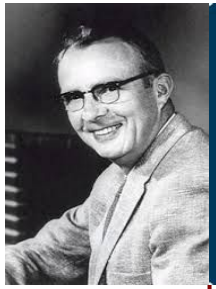
Department of Computer Engineering
Koç University

Didem Unat?

2006
Graduated from
Boğaziçi University

2012
PhD at
University of California,
San Diego

Didem Unat?



BERKELEY LAB
Lawrence Berkeley National Laboratory

MARIE CURIE ACTIONS

BİLİM AKADEMİSİ

THE ROYAL SOCIETY

2012-2014
Luis Alvarez Postdoctoral Fellowship

Lawrence Berkeley National Laboratory

2014 -
Koç University

KOÇ UNIVERSITY
PARALLEL AND MULTICORE COMPUTING LABORATORY

ParCoreLab




1 Post-doc

6 PhD Students

11 MSc Students

30+ Publications





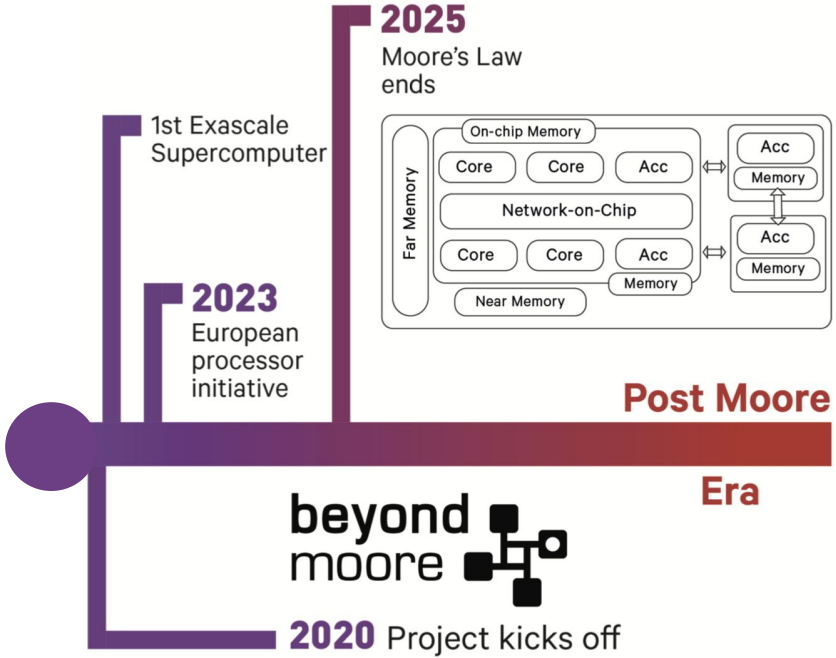
Turkey, Pakistan, Iran, Indonesia, Albania, Kazakhstan, Tanzania, Palestine ...

Collaborations with Universities, Research Labs and Industry

European Research Council

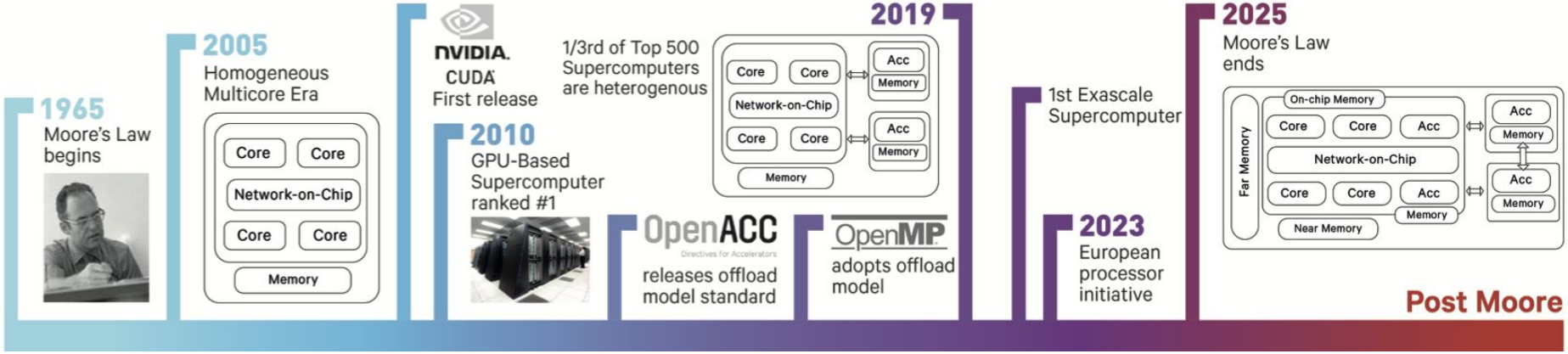
Established by the European Commission

1.5 Million Euros



Supporting top researchers anywhere in the world

First ERC grant received from Turkey in Computer Science



BeyondMoore addresses the most timely and challenging issue of computing.

How to efficiently and productively program future computers in Post-Moore's Era?

Aims to solve the software-side of Post-Moore's crisis

SPARCITY

An Optimization and Co-design Framework for Sparse Computation

2.6 Million Euros

Project Coordinator in an European consortium with 6 partners

This project has received funding from the European High-Performance Computing Joint Undertaking under grant agreement No. 956213.

SparCity aims at creating ***a supercomputing framework*** *that will provide* ***efficient algorithms and coherent tools*** *designed for* ***sparse computations****, while also opening up new usage areas in graph analytics.*

KOÇ UNIVERSITY

Sabancı Üniversitesi

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

MNM TEAM MUNICH NETWORK MANAGEMENT TEAM

inesc id lisboa

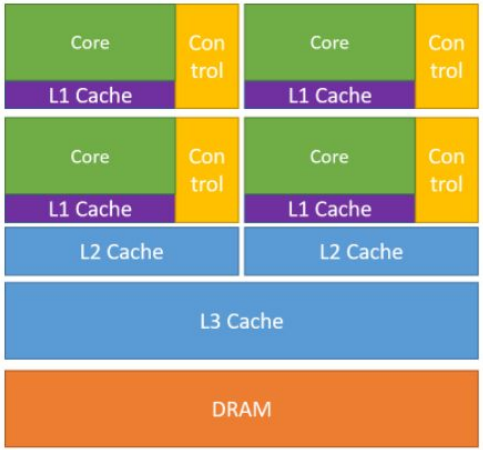
GRAPHCORE

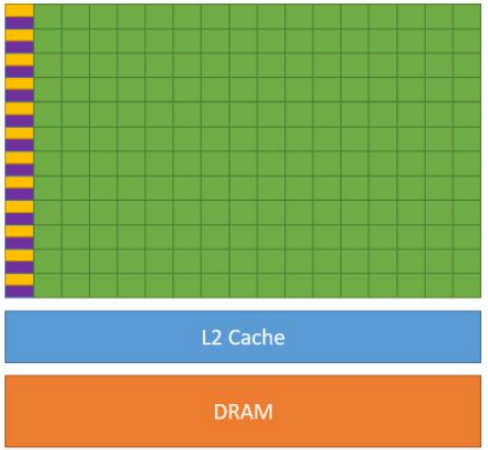
[simula . research laboratory]

Yapay Zeka için HPC Çözümleri

DL Needs Throughput-Oriented Architecture

- DL models are compute intensive
- GPUs played major role in the renaissance of DL
  - Order of magnitude faster training
  - Many cores
  - High bandwidth memory





Memory Bottleneck

- Accelerators (GPUs) have a limited device memory
  - GPU V100 comes with 32 GBs
  - Technology limitations and price

Memory Bottleneck

- Accelerators (GPUs) have a limited device memory
  - GPU V100 comes with 32 GBs
  - Technology limitations and price

- DNNs grow in size
  - Higher accuracy on more complex tasks (Transformers)
  - Faster training
    - Wide ResNet vs ResNet
    - WRN-16-8 >> ResNet-101

Memory Bottleneck

- Accelerators (GPUs) have a limited device memory
  - GPU V100 comes with 32 GBs
  - Technology limitations and price

- DNNs grow in size
  - Higher accuracy on more complex tasks (Transformers)
  - Faster training
    - Wide ResNet vs ResNet
    - WRN-16-8 >> ResNet-101

- Models barely fit into single GPU memory
  - Use small batch sizes
    - Resource underutilization
- Models do not fit into single GPU memory

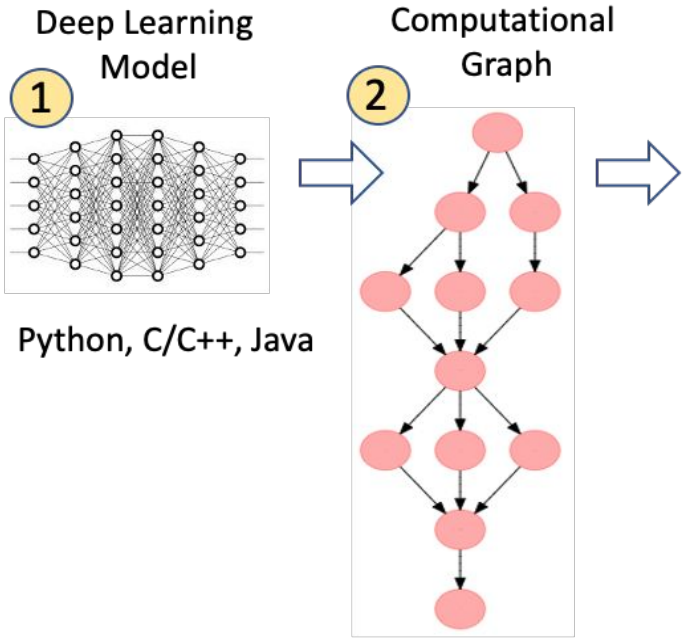
Related Work

- (1) Single device based solutions
  - Memory optimization techniques (Gradient Checkpointing)
  - Utilizing the host memory (Unified Memory)
- (2) Distributed training
  - Data parallelism
    - Doesn’t address the memory issue
  - Model parallelism (Gpipe, Pipedream, and others)
    - Model-specific, not general
    - Accuracy issues, requires manual tuning/implementations
  - Hybrid parallelism (Mesh-TensorFlow)
    - Specific, requires manual tuning

Our Approach: ParDNN

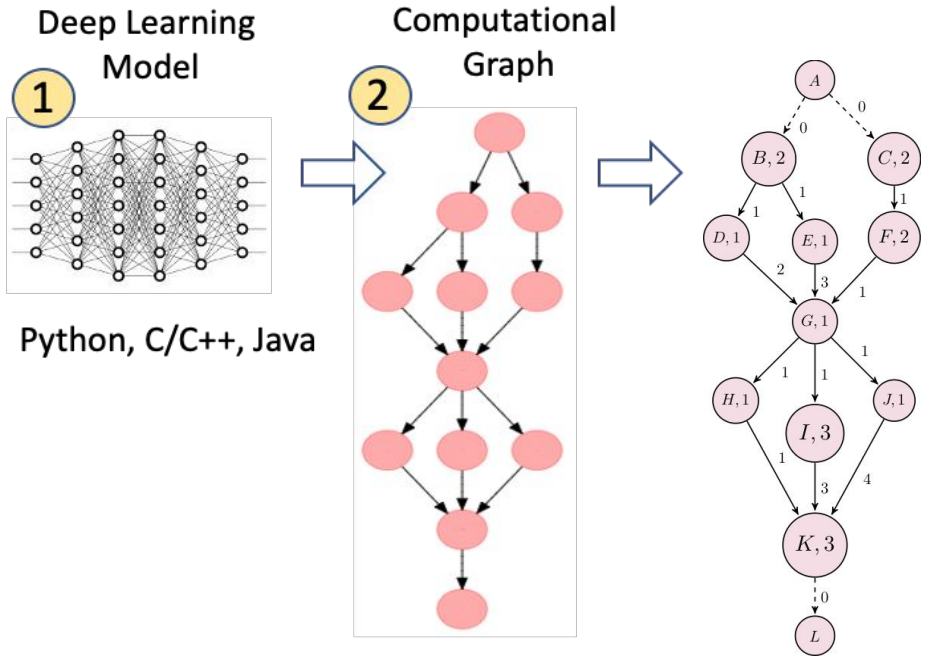
- Generic
  - Zero dependency and requires no knowledge about the DL aspects of the DNN models
- Automated, non-intrusive
  - Requires no modification of the model or operation kernels
- Works at system-level
  - Operates on computational graph

Computational Graph



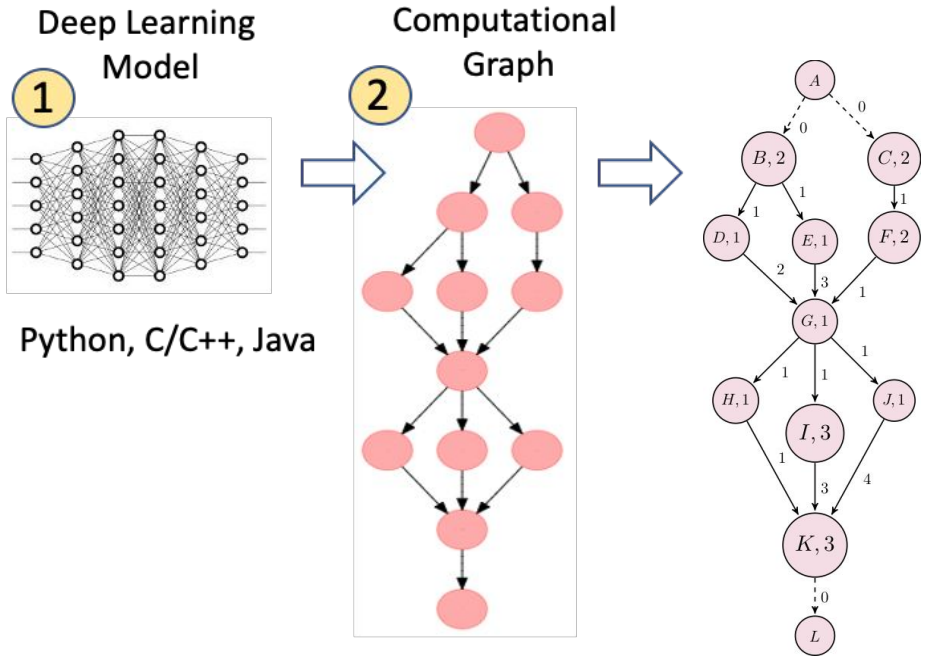
- Operations in the graph represent one step
  - Both forward pass and back propagation are in the graph
- The graph is **static**
  - Constructed before running and stays the same
  - There are dynamic cases
- The graph is acyclic

Computational Graph



- *G(V,E)*: Task graph
- *V*
  - *n* ∈ *V*: Task.
  - *w(n)*: weight of n, computation time
- *E*
  - *e* ∈ *E*: Dependency.
  - *c(e):* cost of e, communication time
  - Defines the execution order

Computational Graph



- *G(V,E)*: Task graph
- *V*
  - *n* ∈ *V*: Task.
  - *w(n)*: weight of n, computation time
- *E*
  - *e* ∈ *E*: Dependency.
  - *c(e):* cost of e, communication time
  - Defines the execution order

How to partition this task graph among multiple GPUs?

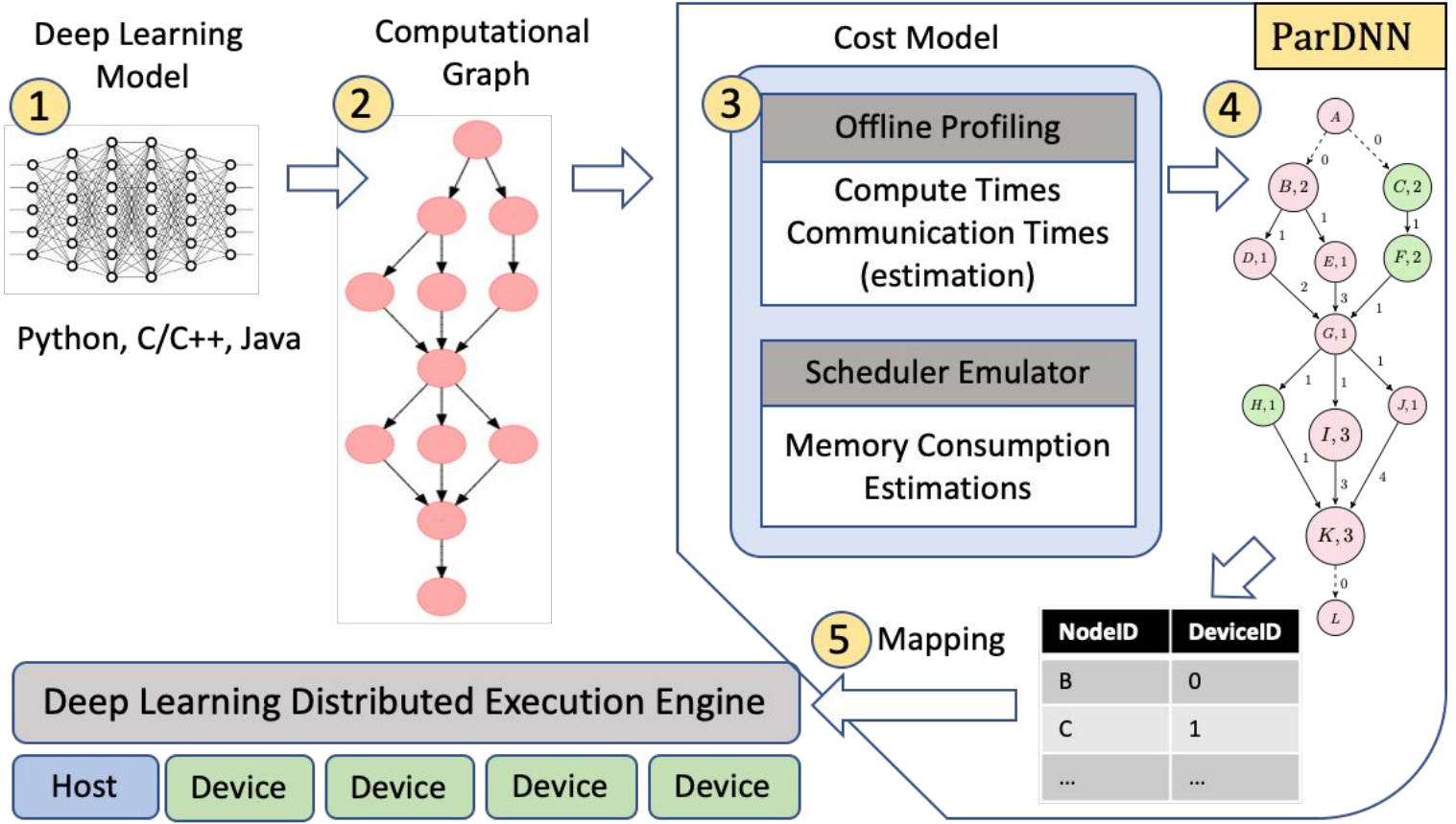
- *obey the memory constraints,*
- *reduce communication,*
- *minimize execution time*

Real DNN Graphs

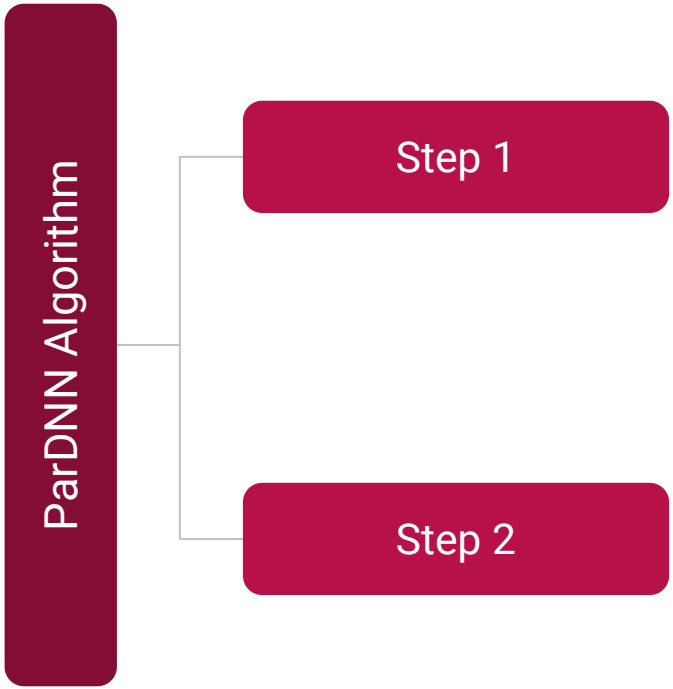
- Number of operations reaches hundreds of thousands, may scale up to millions.
  - **Another objective**: Low complexity is necessary

Model	Acronym	#Layers	HSD	SL	#Parameters	#Graph Nodes	Dataset
Recurrent Neural Network for Word-Level Language [51]	Word-RNN	10	2048	28	0.44 billion	11744	Tiny Shakespeare [23]
	Word-RNN-2	8	4096	25	1.28 billion	10578	
		#Layers	CHSD	ED			
Character-Aware Neural Language Models [26]	Char-CRN	8	2048	15	0.23 billion	22748	Penn Treebank (PTB) [33]
	Char-CRN-2	32	2048	15	1.09 billion	86663	
		#Conv. Layers [65]	#RU	WF			
Wide Residual Network [64]	WRN	610	101	14	1.91 billion	187742	CIFAR100 [28]
	WRN-2	304	50	28	3.77 billion	79742	
		#Layers	HSD	MD			
Transformer [54]	TRN	24	5120	2048	1.97 billion	80550	IWSLT 2016 German–English corpus [6]
	TRN-2	48	8192	2048	5.1 billion	160518	
		#Hidden Layers	FS				
Eidetic 3D LSTM[58]	E3D	320	5		0.95 billion	55756	Moving MNIST digits [50]
	E3D-2	512	5		2.4 billion	55756	

Our Approach: ParDNN

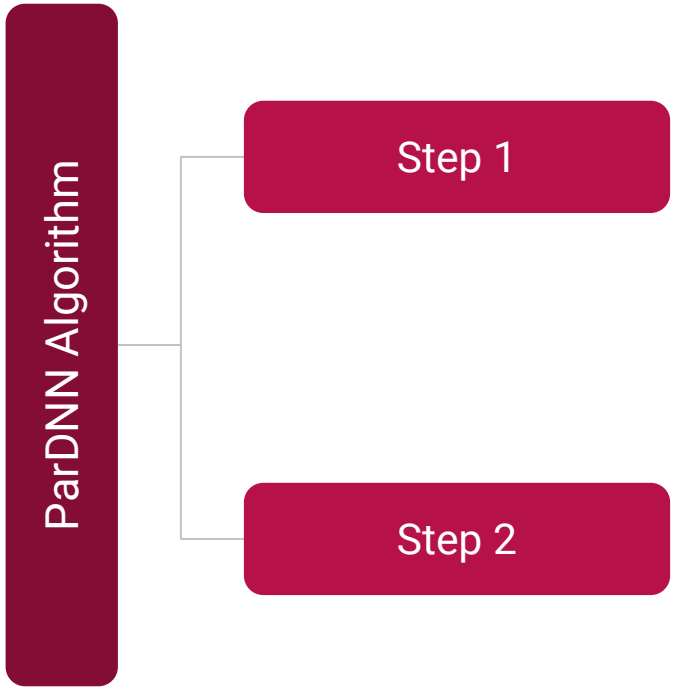


ParDNN Algorithm Overview



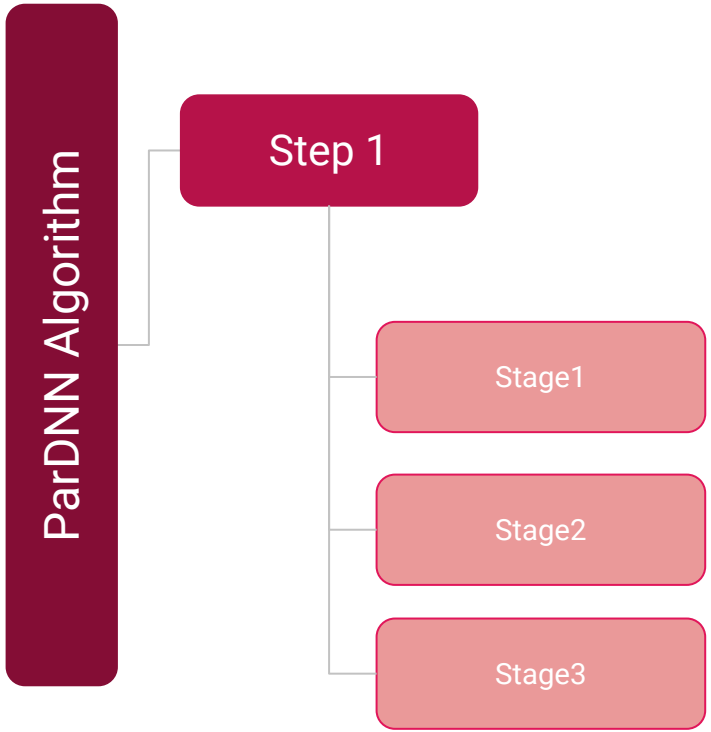
- Step 1: Given *K* devices, partition the graph into *K* partitions so that execution time is minimized
  - Communication time is minimized
  - Computation loads are balanced

ParDNN Algorithm Overview



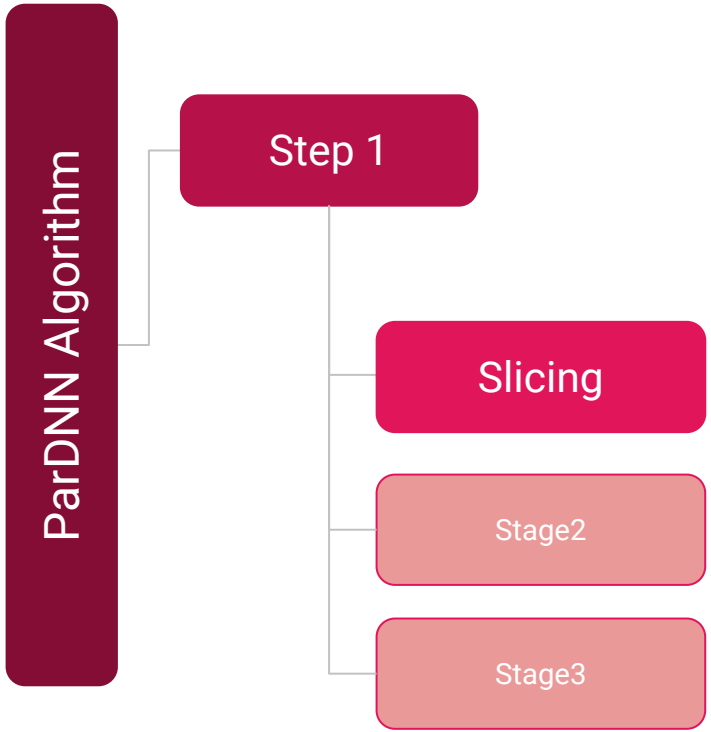
- Step 1: Given *K* devices, partition the graph into *K* partitions so that execution time is minimized
  - Communication time is minimized
  - Computation loads are balanced
- Step 2: Meet the memory consumption constraints
  - If each partition meets the device memory constraints
    - Done.
  - Else
    - Handle the memory overflow while maintaining locality-parallelism trade-off.

ParDNN Algorithm



- To achieve both
  - Good quality partitions
  - Reasonable runtime
- Step 1 is divided into 3 stages

ParDNN Algorithm

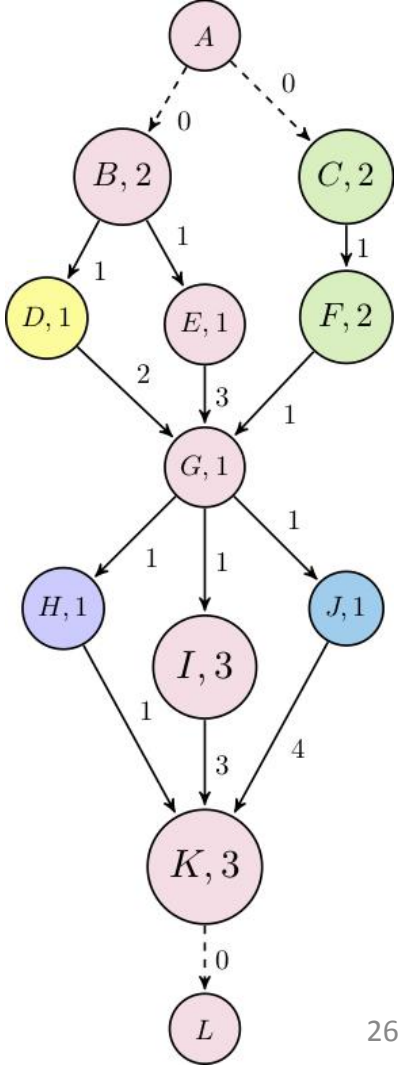


- To achieve both
  - Good quality partitions
  - Reasonable runtime
- Step 1 is divided into 3 stages:
  - Stage 1: Slicing
    - Gets smaller instance representation
    - Obtaining coarser view
    - Capturing costly communications

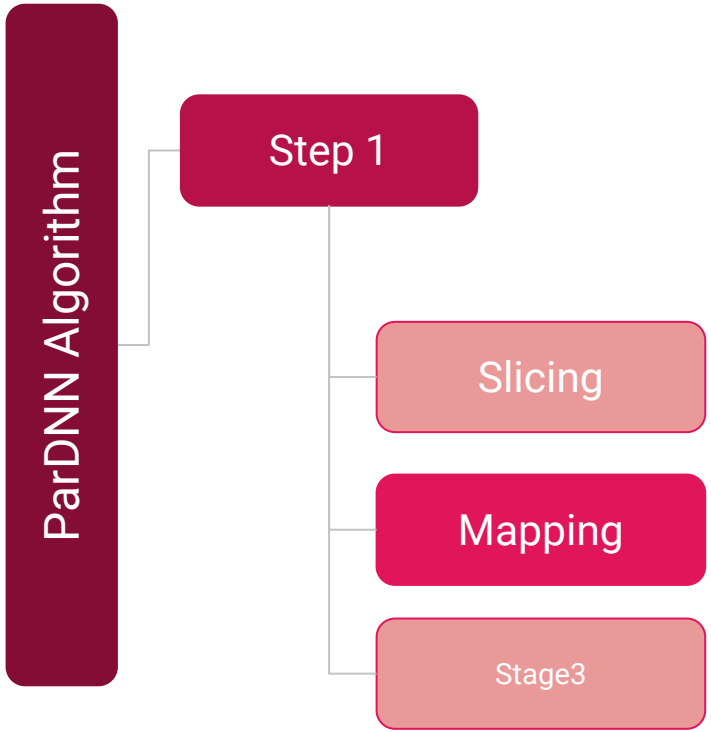
Graph Slicing

- Obtain *K* critical paths of the graph
  - Get the critical path
    - **primary cluster**
  - Remove its nodes & incident edges
- Until the graph has no more nodes
  - Find the heaviest cluster
    - **secondary cluster**
  - Remove its nodes & incident edges

In the figure, pink and green paths are primary clusters
Yellow, blue and purple nodes are secondary clusters



ParDNN Algorithm Overview

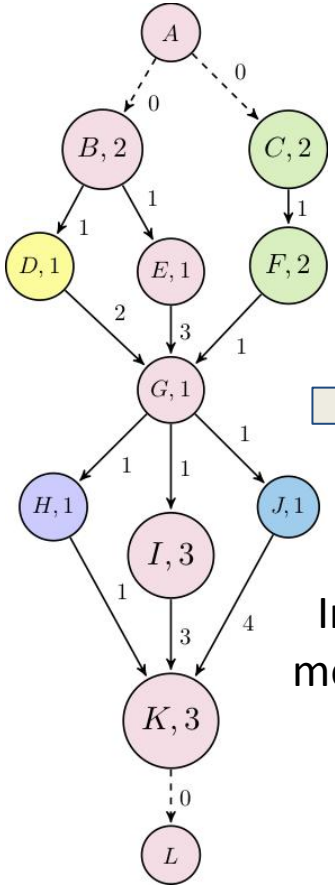


- To achieve both
  - Good quality partitions
  - Reasonable runtime
- Step 1 is divided into 3 stages:
  - Stage 1: Slicing
  - Stage 2: Mapping, **merge secondary clusters with primaries** in a way that:
    - Balances computational loads
    - Minimizes communication

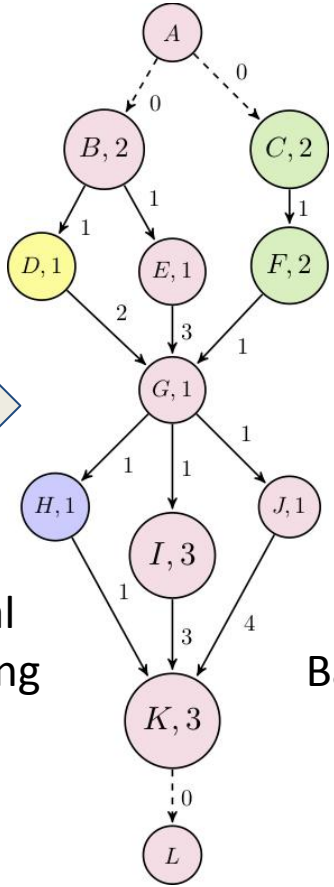
Mapping

- Initial merging
  - Merges secondary clusters that have no parallelism gain
- Level-aware load balancing
  - Pick a secondary cluster and merge it with one on the primaries such that
    - This primary has the least load within its span
    - The incurred communication is minimized.

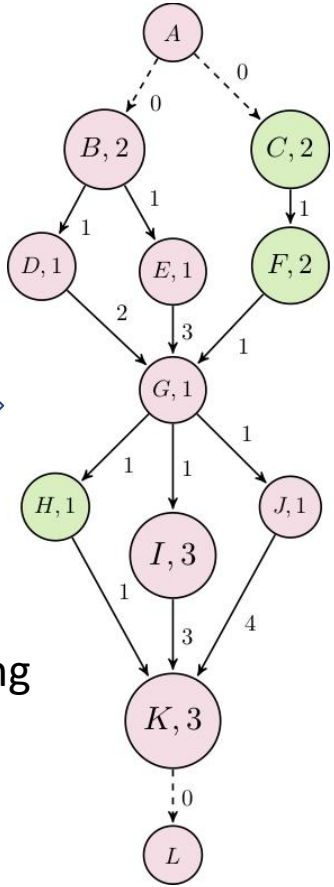
Mapping



Initial merging

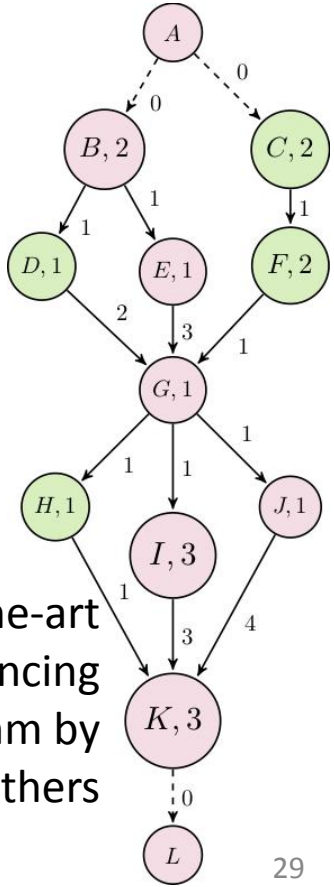


Load Balancing

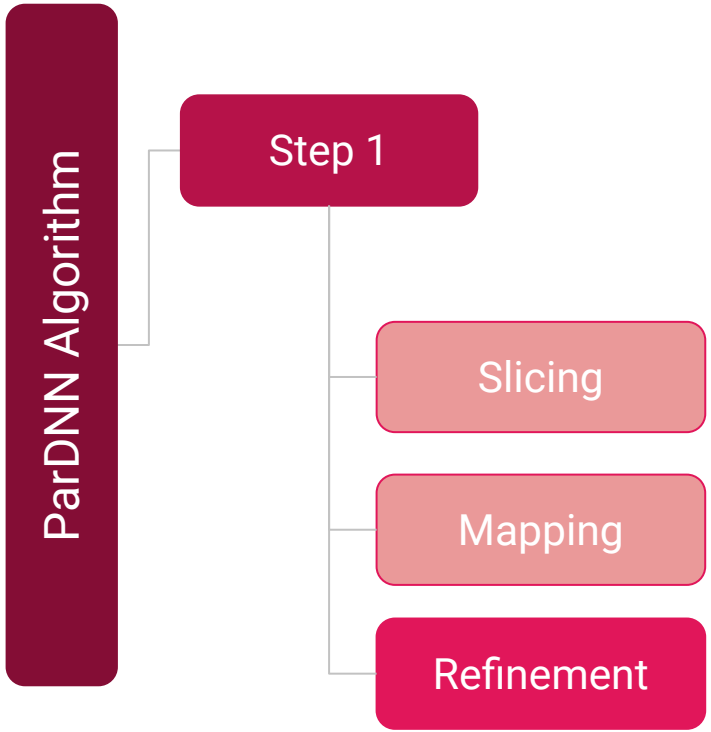


VS

State-of-the-art load balancing algorithm by others



ParDNN Algorithm Overview

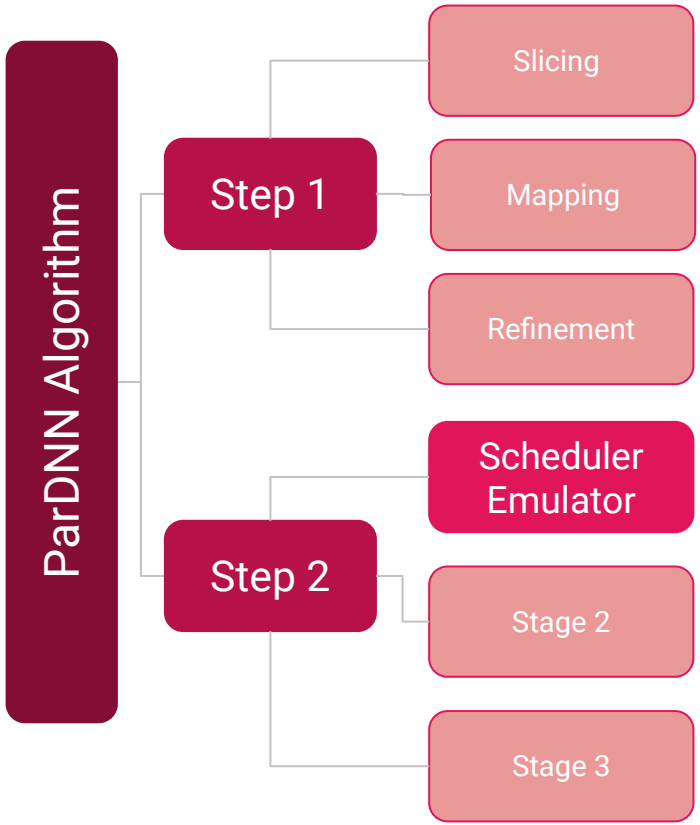


- To achieve both
  - Good quality partitions
  - Reasonable runtime
- Step 1 is divided into 3 stages:
  - Stage 1: Slicing
  - Stage 2: Mapping
  - Stage 3: Refinement
    - Enhance partitioning quality
      - At the cluster level
      - At the node level
    - Swap paths and nodes between primaries

Refinement

- Path swapping:
  - Swap a cluster *c* with:
    - *c'* within the span of *c*, such that swapping *c* with *c'* improves the quality.
- Node switching:
  - Focus on communication edges
  - When there are many heavy communications:
    - Some may fall outside the clusters, thus participate in creating heavy critical paths.
  - In newly formed critical path.
    - Switch such node placements if this shortens it.

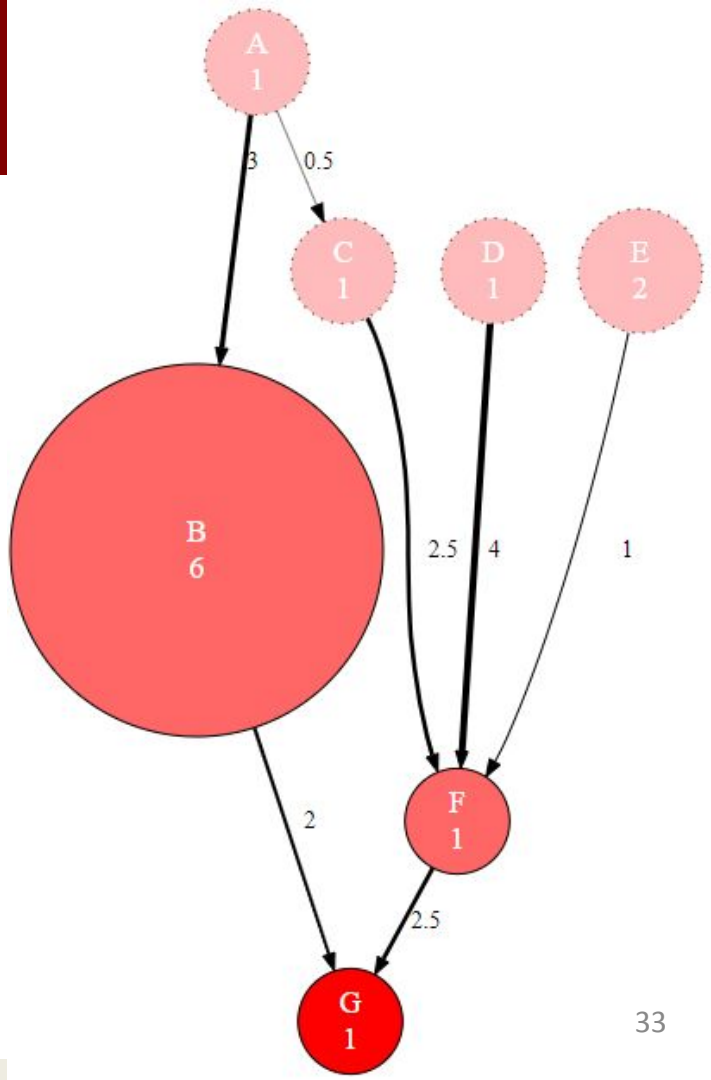
Step 2: Meeting Memory Constraints



- Step 2: Meet the memory consumption constraints
  - Stage 1: Emulate Tensorflow scheduler
    - Get the node's expected **scheduling times**
    - Memory allocation and deallocation patterns

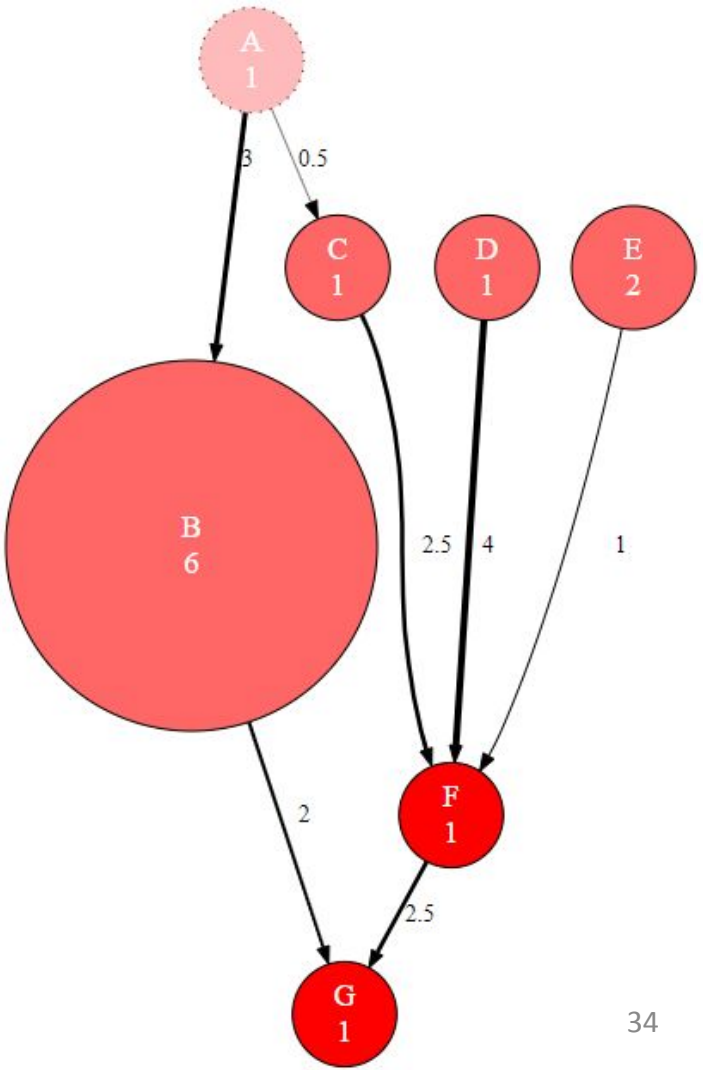
Memory consumption

- Assume a schedule:
  - A, C, D, E, F, B, G.
- Peak memory reserved:
  - 1 + 6 + 1
    - = 8

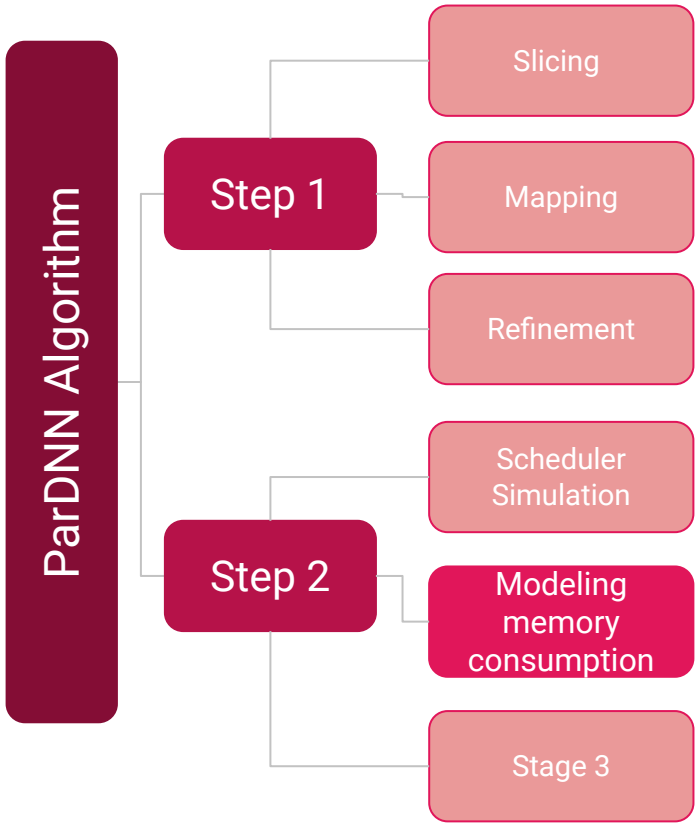


Memory consumption

- Assume **another** schedule:
  - A, B, C, D, E, F, G.
- Peak memory reserved:
  - 6 + 1 + 1 + 2 + 1 = **11**
- It affects as well when having multiple workers.
  - When the data is sent from one to another.

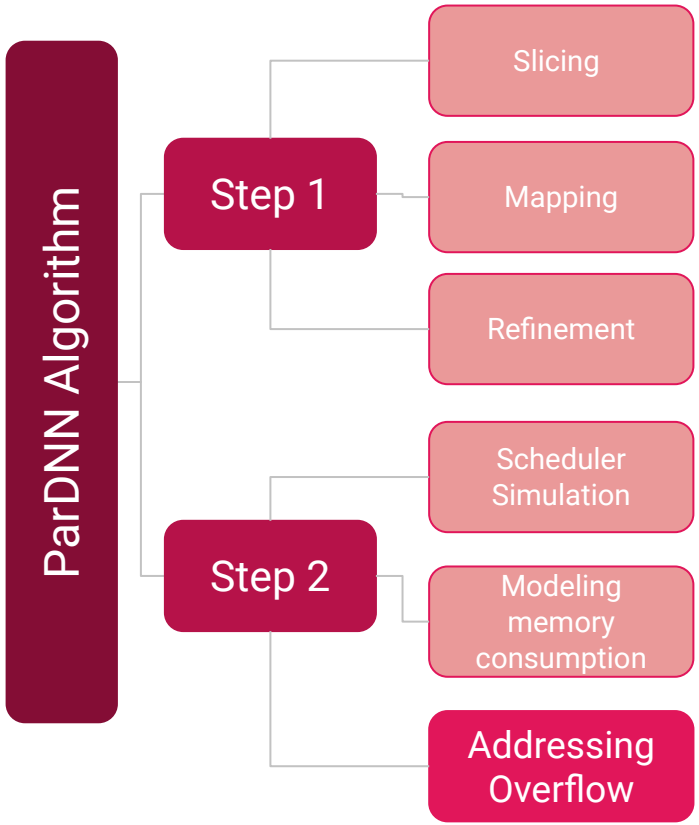


Step2 Stages



- Stage 1: Emulate Tensorflow scheduler
- Stage 2: Modeling memory consumption
  - Derive the memory consumption on a certain device at a certain point in time
  - Calculate memory potentials

Step2 Stages



- Stage 1: Emulate Tensorflow scheduler
- Stage 2: Modeling memory consumption
- Stage 3: Address the memory overflow
  - Which nodes to move?
  - Where to move?

Addressing Memory Overflow

- Each overflow point can be 0-1 min knapsack
  - Move a set of nodes from the overloaded part
    - Summation of their memory potentials at the overflow time ≥ Overflow
  - The cost of a move is how much it affects the existing partitioning:
    - Incur the least possible perturbation on Step 1 results
  - Solved greedily
  - Move the node which , per a memory unit, has the least computation cost and incurs the least communication when moved.

Results

Models and Datasets

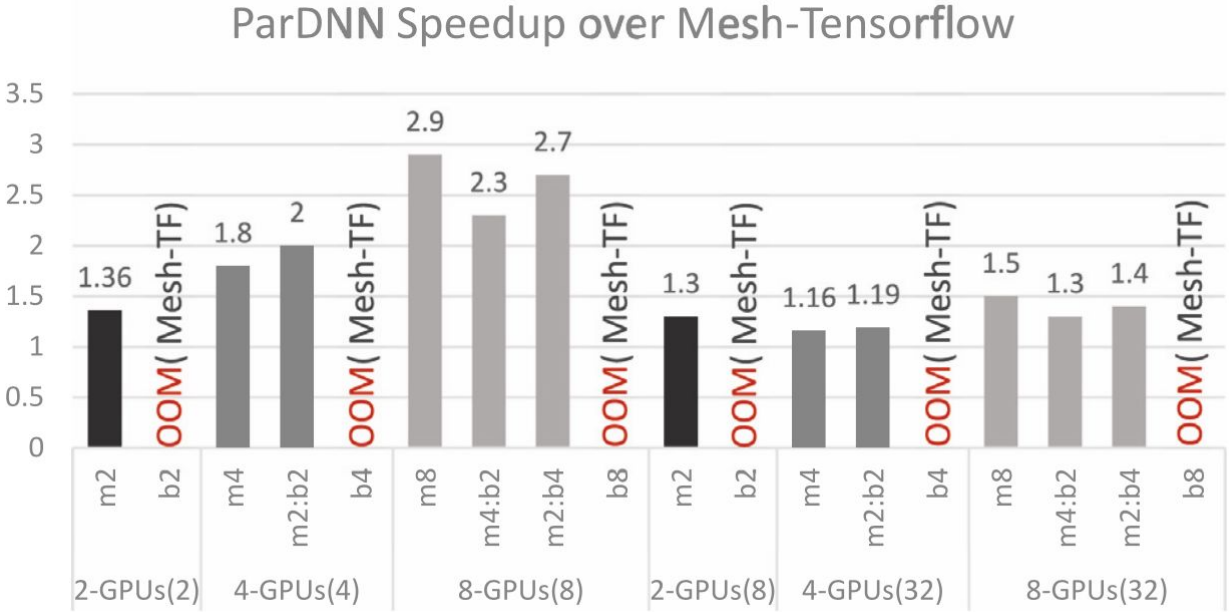
- We have experimented with 5 models with 2 different configurations (large and very large)

Table 3
Specifications of models datasets.

Model/Dataset	Acronym	#Layers	HSD	SL	#Para. (10^9)	#Graph nodes
RNN for Word-Level Language [38]/Tiny Shakespeare [39]	Word-RNN	8	2048	28	0.34	10578
	Word-RNN-2	32	2048	25	1.18	39074
			CHSD	ED		
Character-Aware Neural Language Models [40]/Penn Treebank (PTB) [41]	Char-CRN	8	2048	15	0.23	22748
	Char-CRN-2	32	2048	15	1.09	86663
			#RU	WF		
Wide Residual Net. [5]/CIFAR100 [42]	WRN	610	101	14	1.91	187742
	WRN-2	304	50	28	3.77	79742
			HSD	MD		
Transformer [43]/IWSLT'16 German–English corpus [44]	TRN	52	4098	2048	1.99	204792
	TRN-2	48	8192	2048	5.1	160518
		HSD	FS	P_SZ		
Eidetic 3D LSTM [45]/Moving MNIST digits [46]	E3D	320	5	4	0.95	55756
	E3D-2	512	5	8	2.4	55756

(C)HSD: (Character) Hidden State Dimension, SL: Sequence Length, ED: Embedding Dimensions, RU: Residual Units, WF: Widening Factor, MD: Model Dimension, FS: Filter Size, P_SZ: patch size.

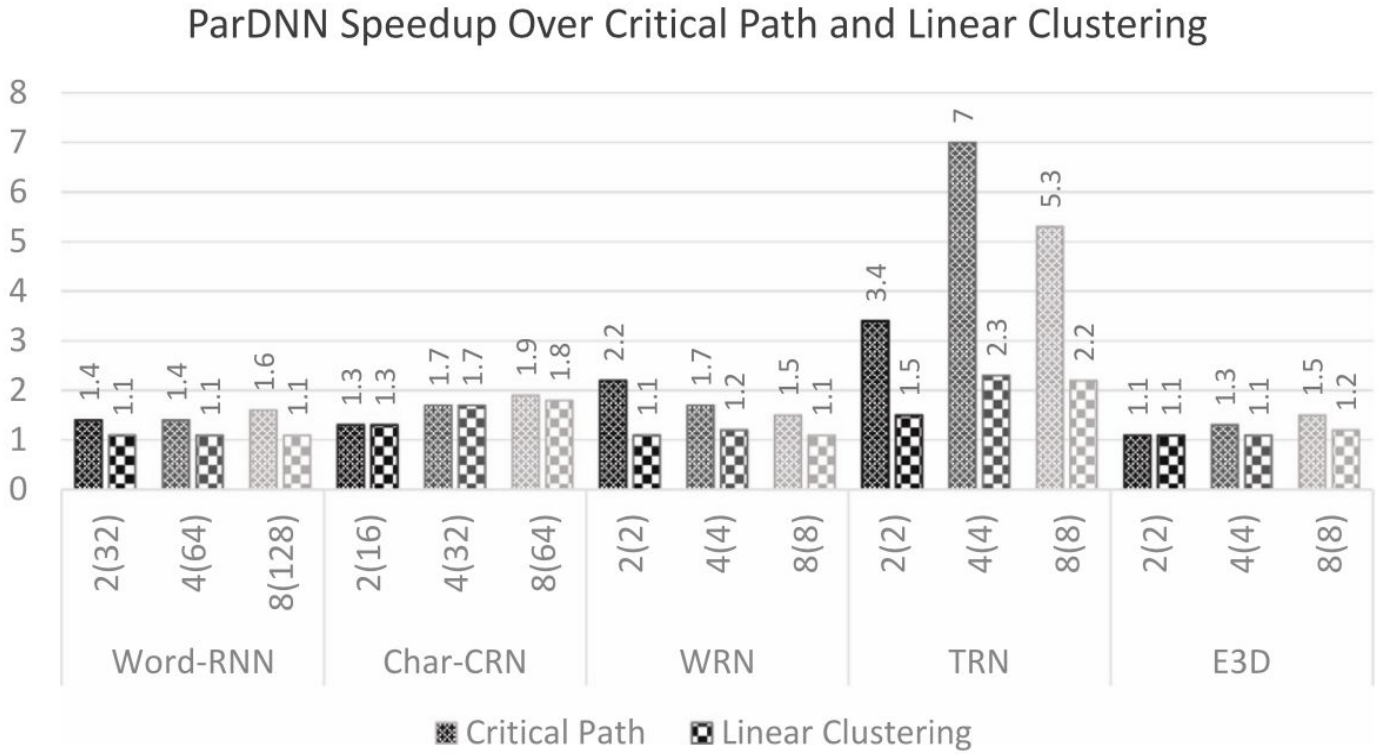
Comparison with Mesh-TensorFlow



The number in the brackets is the batch size.

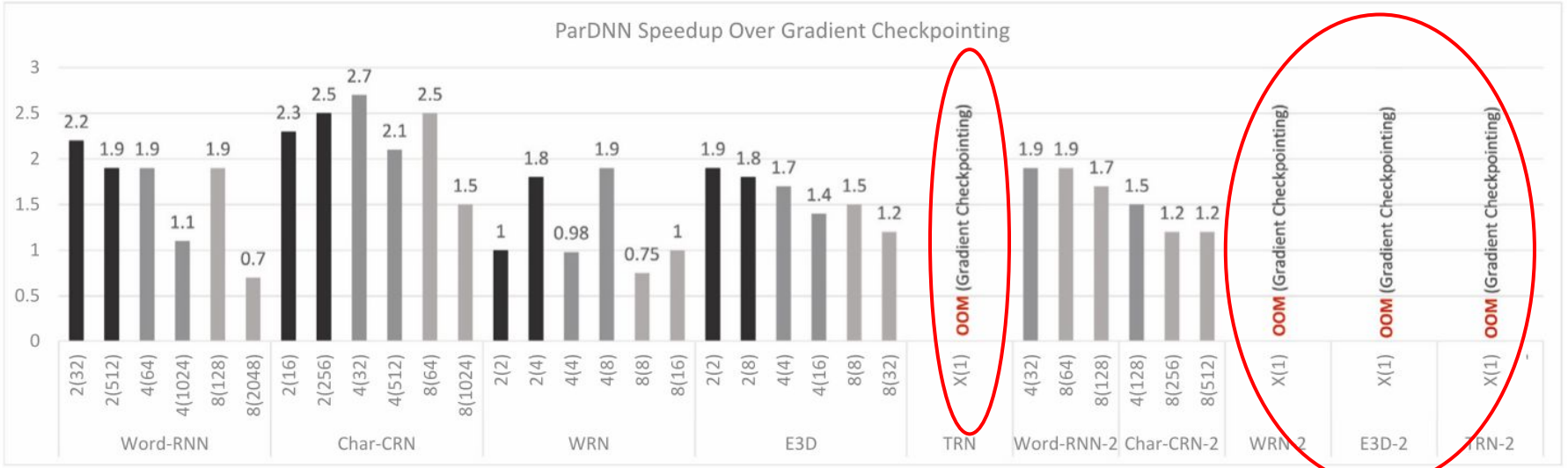
Speedup over Mesh-TensorFlow developed by Google using Transformer Model

Comparison with Critical Path Methods



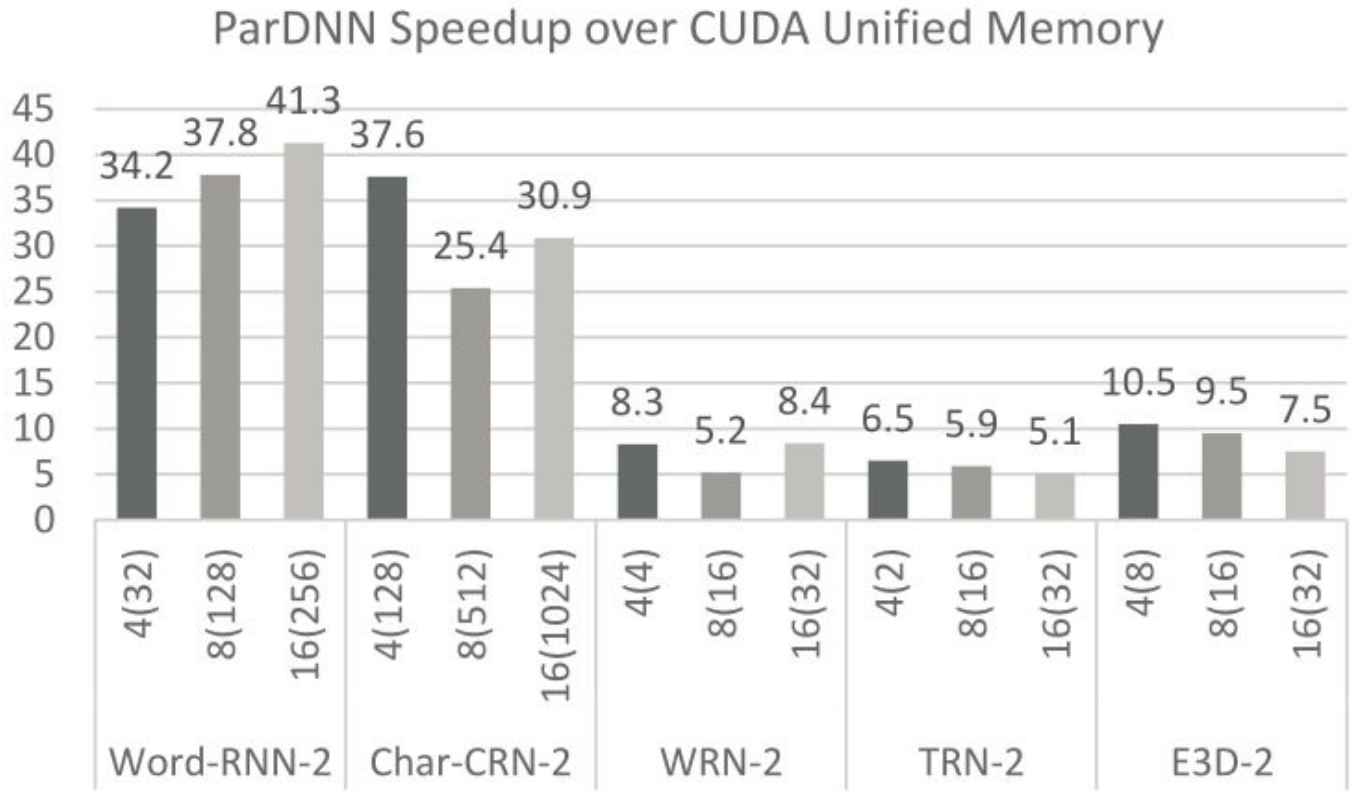
R. Mayer, C. Mayer, L. Laich, The tensorflow partitioning and scheduling problem: It's the critical path! in: Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning, 2017, pp. 1–6.

Comparison with Gradient Checkpointing

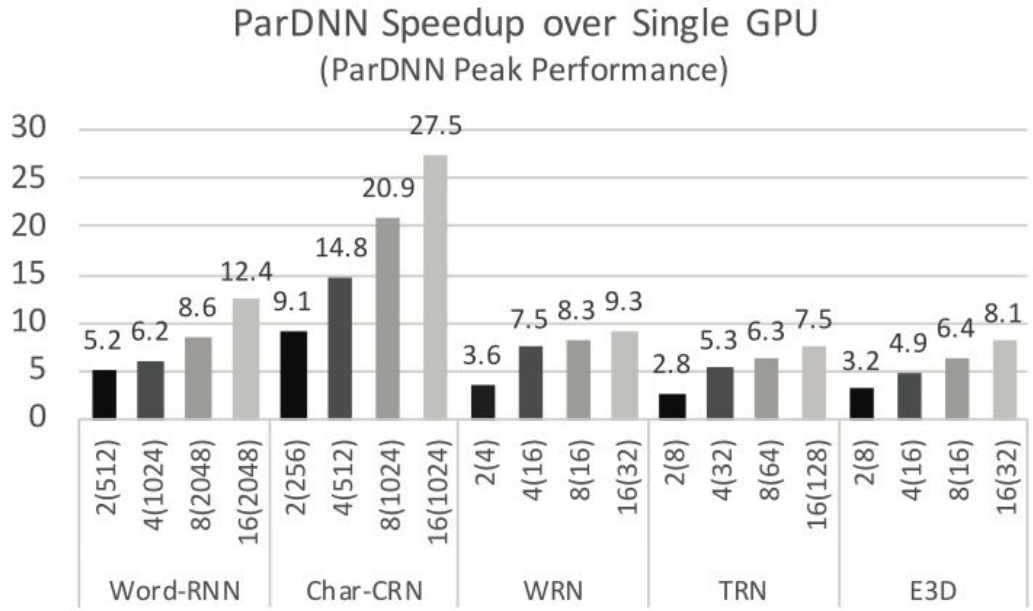


- ParDNN is better in most of the cases.
- Checkpointing Fails to fit the model in some cases

Comparison with UVM



Results (Training Speedup)



Better resource utilization → Superlinear speedup up to 4 GPUs in all cases.

Results (Batch size scaling)

	Batch size scaling				
Model/#GPUs	1	2	4	8	16
Word-RNN	16	512	1024	2048	2048
Char-CRN	8	256	512	1024	2048
WRN	1	4	16	16	32
TRN	1	8	32	64	128
E3D	1	8	16	16	32
Word-RNN-2	–	–	32	128	256
Char-CRN-2	–	–	128	512	1024
WRN-2	–	–	4	16	32
TRN-2	–	–	2	16	32
E3D-2	–	–	8	16	32

ParDNN enables working with larger data, e.g. pushing larger batches, using certain number of workers.

Time Complexity

The running time of our algorithm in all the experiments ranges from 18 to 117 sec

Table 1
Complexity of Each Step of PARDNN.

Step-1	Partition to minimize makespan
Graph slicing (inc. sorting)	$O(K(\|V\| + \|E\|))$
Mapping	$O(\|V\| * \log^2 \|V\|)$
Step-2	Memory heuristic - I
TensorFlow scheduler emulator	$O(\|V\| + \|E\|)$
Memory consumption tracker	$O(\|V\|)$
Overflow handler	$O(\|V^2\|)$
Step-2	Memory heuristic-II
Residual Nodes movement and CP splitting	$O(\|V\|)$
Overall PARDNN Complexity (w. Heuristic-I)	$O(\|V\|^2)$
Overall PARDNN complexity (w. Heuristic-II)	$O(\|V\| * log^2(\|V\|) + K\|E\|)$

Summary

- We addressed memory constrained DNN models on multiple GPU devices
  - Elegant, non-intrusive and model agnostic approach
  - Two step algorithm design provides efficiency and low overhead
  - Compared to similar approaches, our results are better or provides qualitative advantages
  - Paper is on arxiv: https://arxiv.org/abs/2008.08636
  - Published in Elsevier Parallel Computing

This project is funded by Tübitak 118E801.

Outlook

Parallel systems are here

- Mostly homogeneous or heterogeneous (CPU + GPU) systems
- As a community we have done a good job in software preparedness for those systems

Post-Moore's Era (2025 onward) will bring more heterogeneity and hardware specialization

- Low precision units, AI units, GPUs, FPGAs, QCs co-exist in a large-scale system.
- Addressing programming issues in those systems will be more challenging.

https://parcorelab.ku.edu.tr/