Dynamic Resource Partitioning for Multi-Tenant Systolic Array Based DNN Accelerator

Midia Reshadi, David Gregg

School of Computer Science and Statistics,

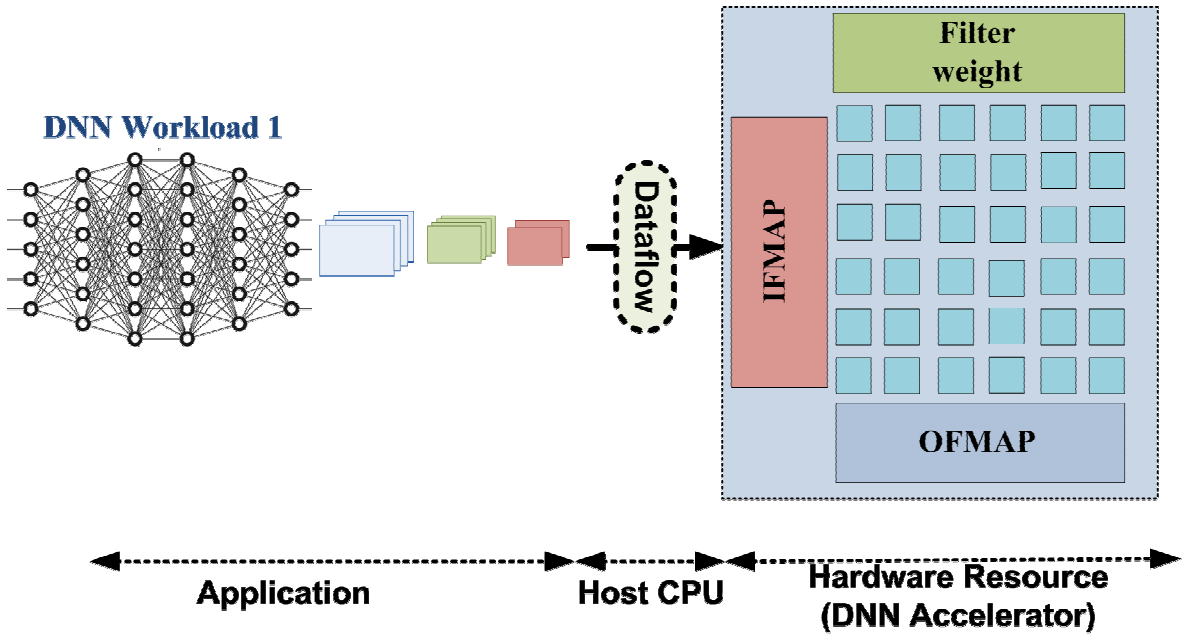
Lero, Trinity College Dublin

Outline

- Introduction
- Preliminaries
- Dynamic Resource Partitioning Algorithm
- LOCAL mapping algorithm
- Simulation Results

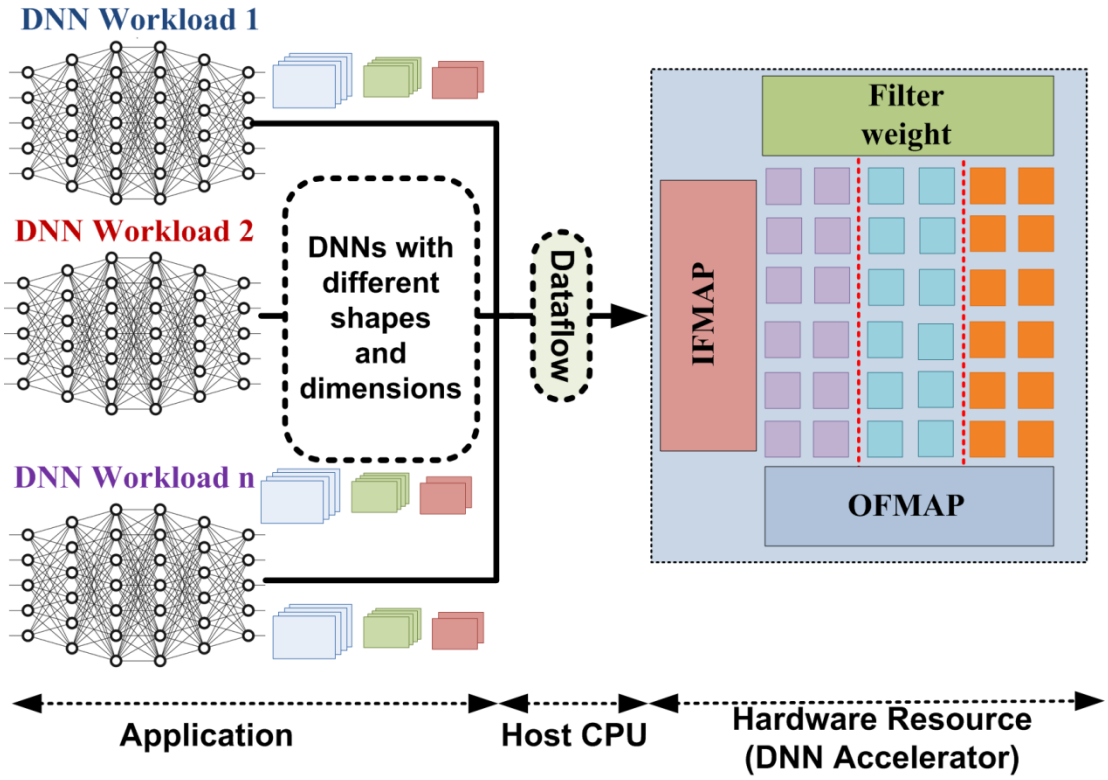
Introduction

- Single tenancy



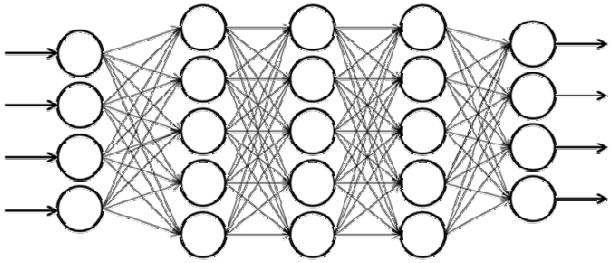
Introduction

- Multi tenancy



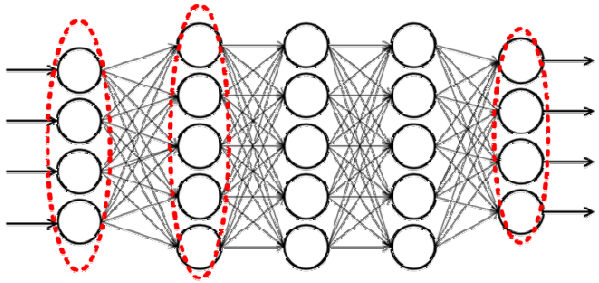
Preliminaries

- Deep Neural Networks Graph (DNNG)



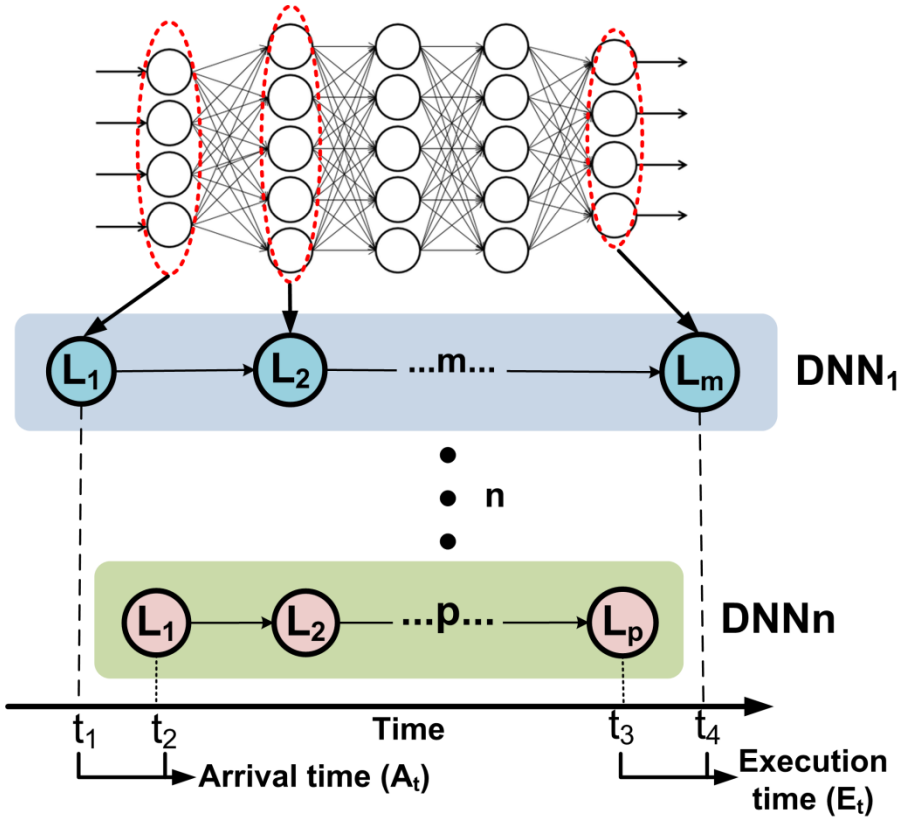
Preliminaries

- Deep Neural Networks Graph (DNNG)



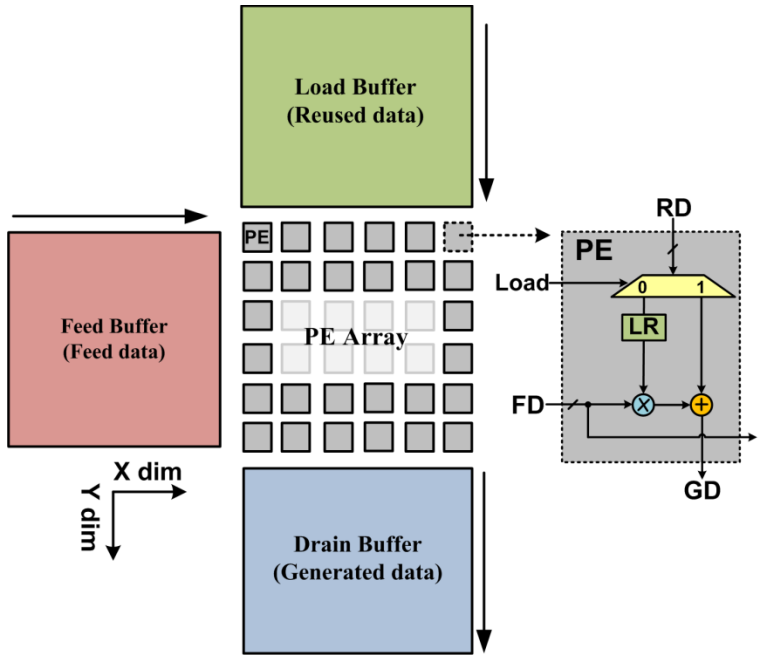
Preliminaries

- Deep Neural Networks Graph (DNNG)



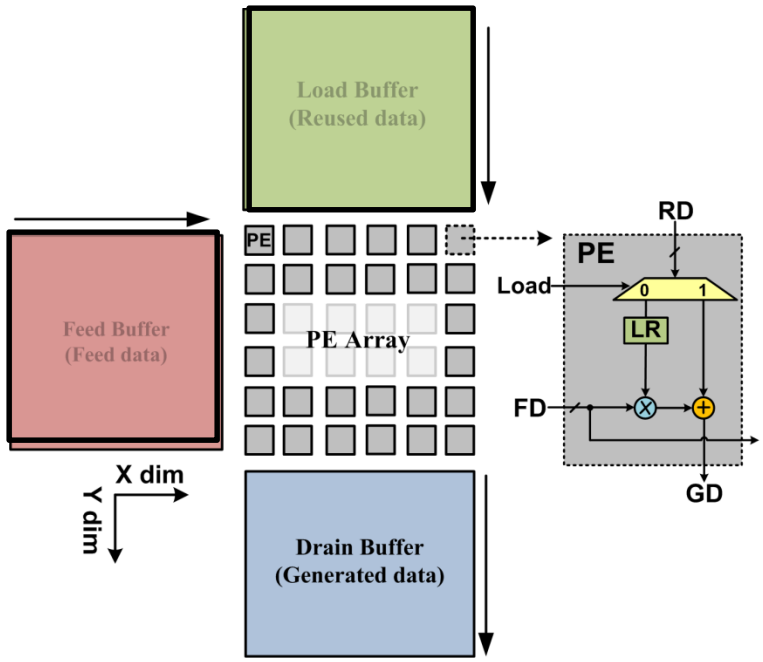
Preliminaries

- Systolic-array Based DNN Accelerator



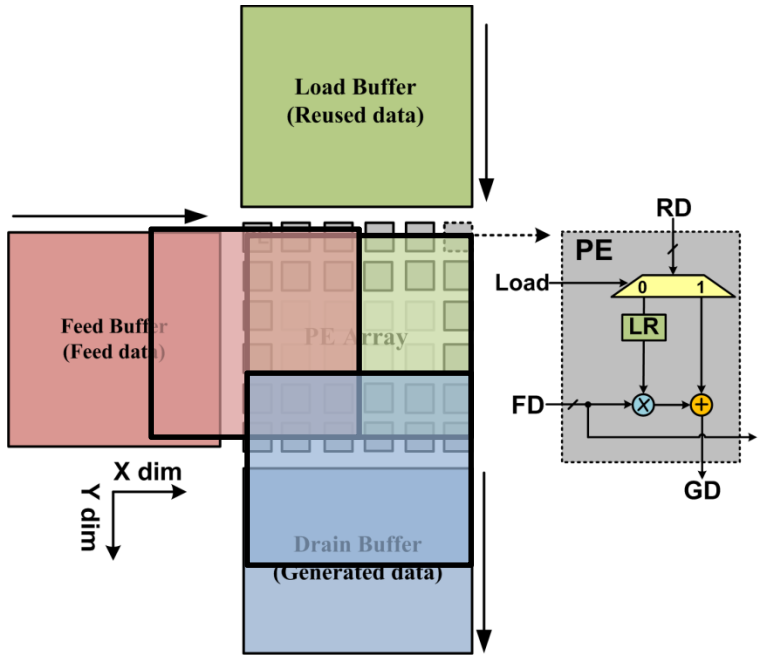
Preliminaries

- Systolic-array Based DNN Accelerator



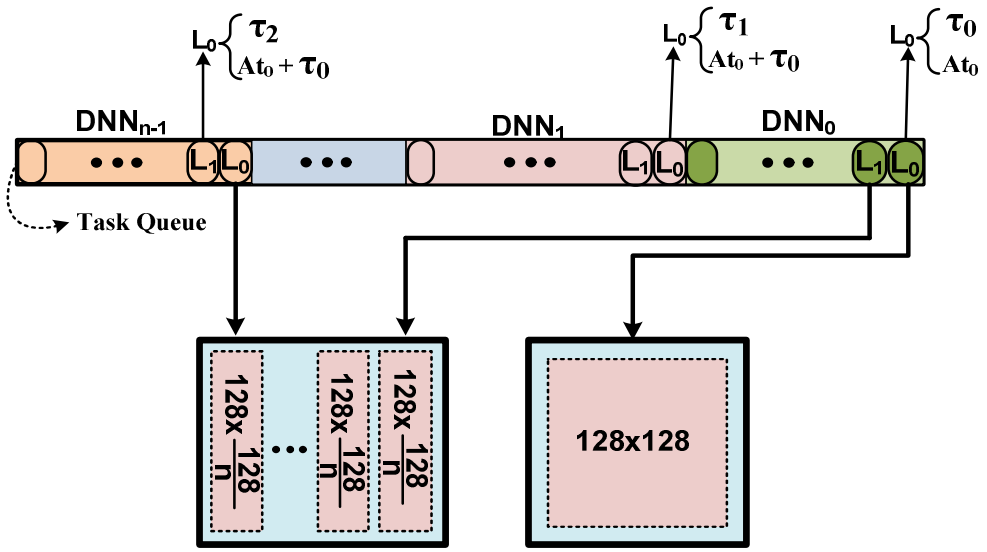
Preliminaries

- Systolic-array Based DNN Accelerator



Dynamic Resource Partitioning Algorithm

- Systolic Array Partitioning



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

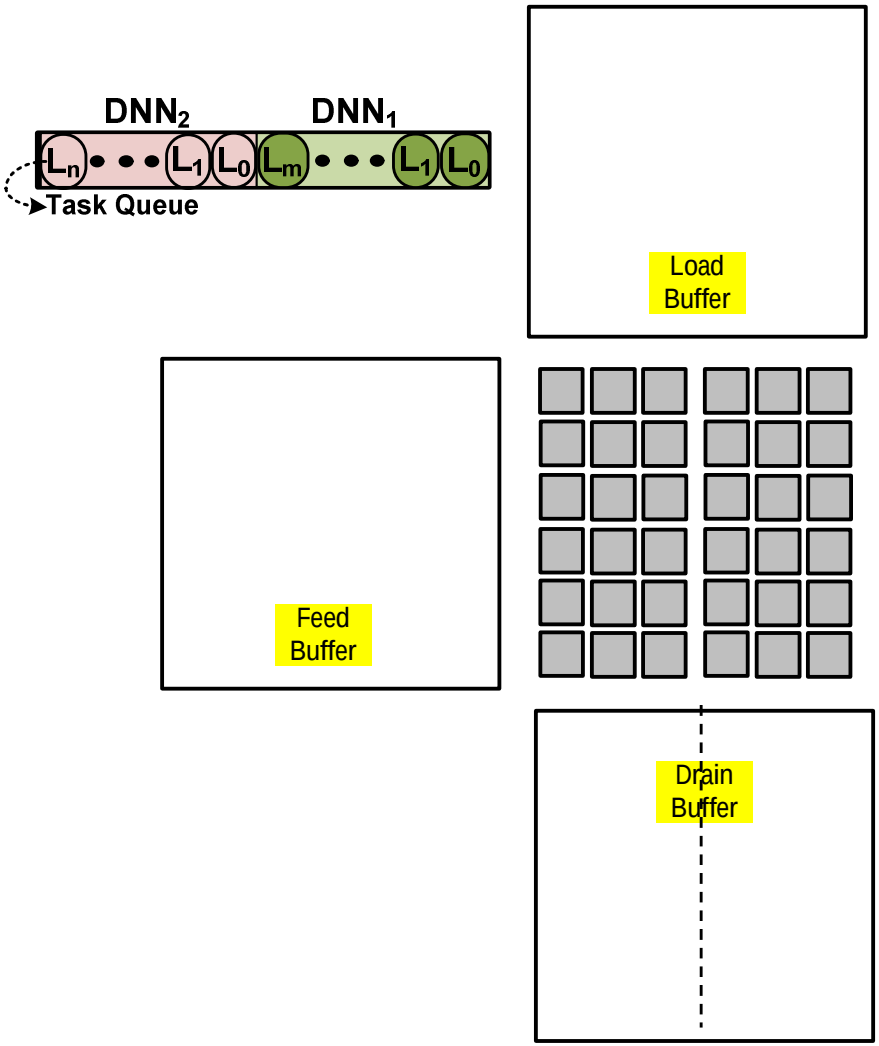
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x,y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

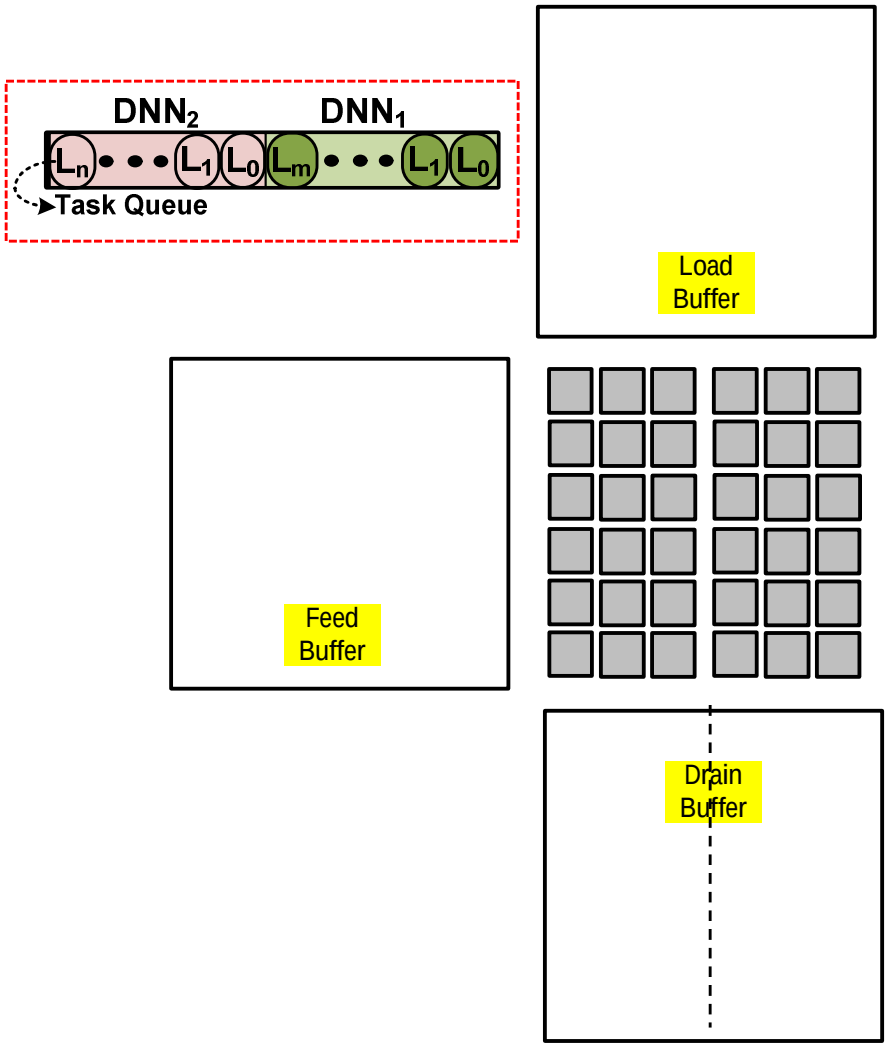
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x*,*y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

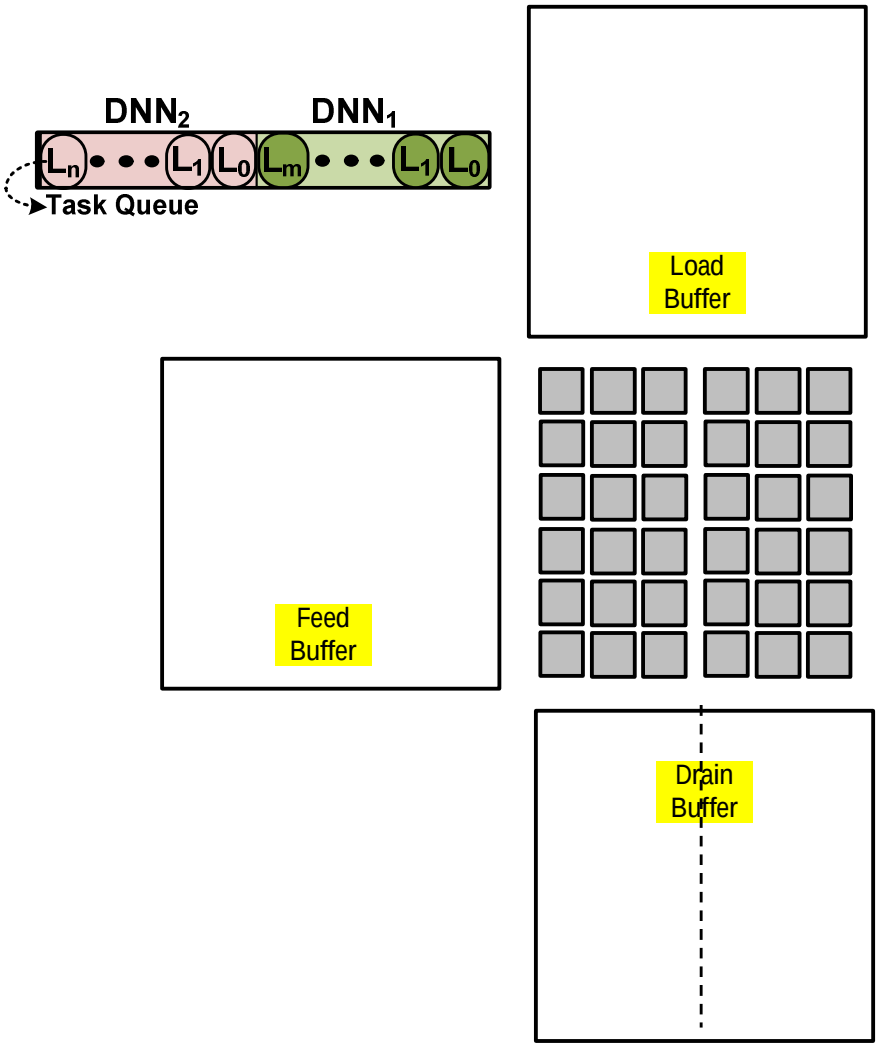
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x,y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

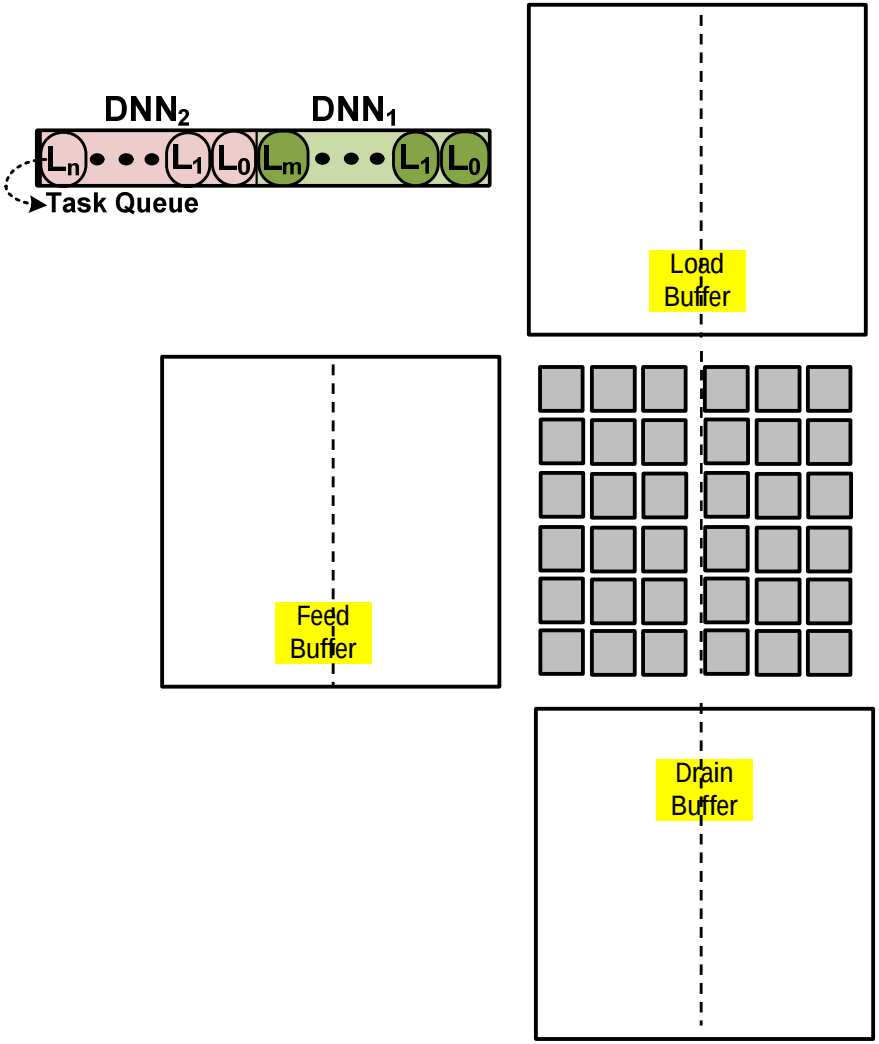
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x,y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
for All available l_i
  Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
  Sort $Opr(l_i)$ from high to low
  Assign l_i with the highest Opr to available partition
end_for
return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
n = Number of available layers
for All partitions n
  //step ❸
  Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
  Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
    //step ❷
    Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
      Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
        //step ❶
        Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
          Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
  end_parallel_temporal_for
end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

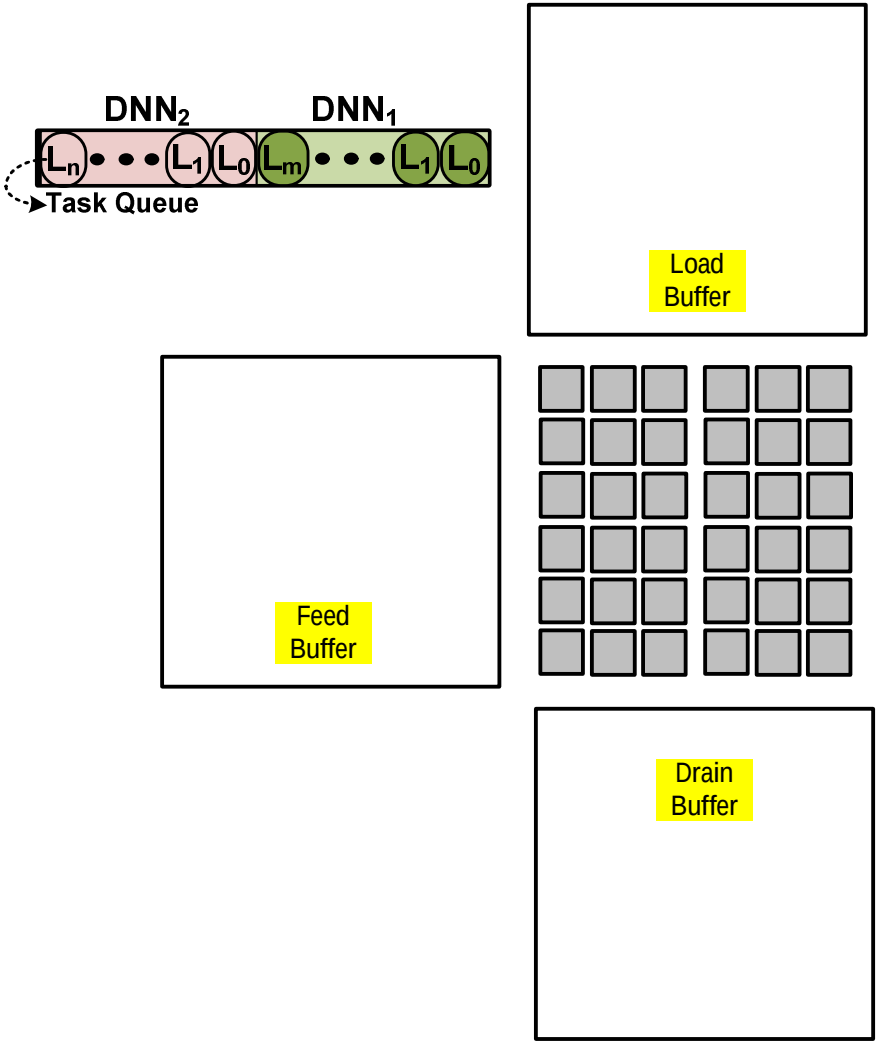
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x*,*y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

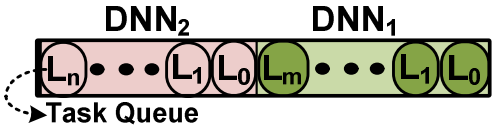
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x*,*y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

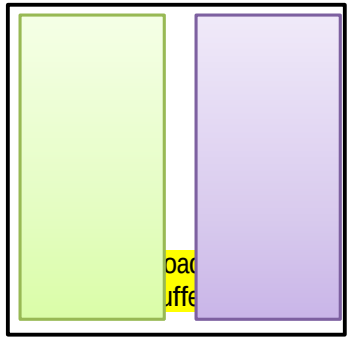
Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

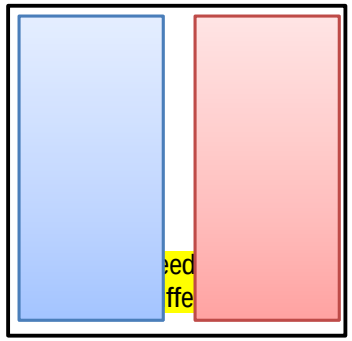
Function Partition_Calculation (*Number of available layers*)
  $PE_{x\prime} = PE_x$ //Partition size in x dimension
  $PE_{y\prime} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

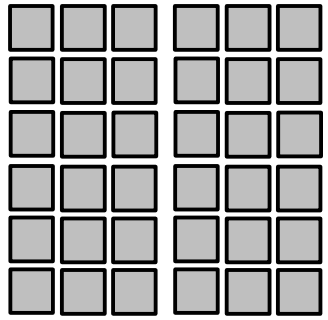
Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

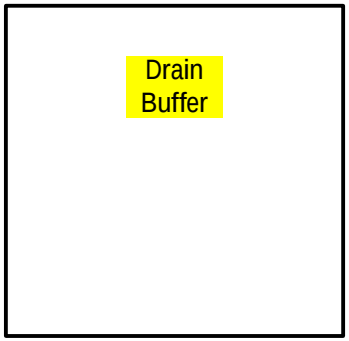
Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
    Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
      //step ❷
      Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
        Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
          //step ❶
          Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
          Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function











Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

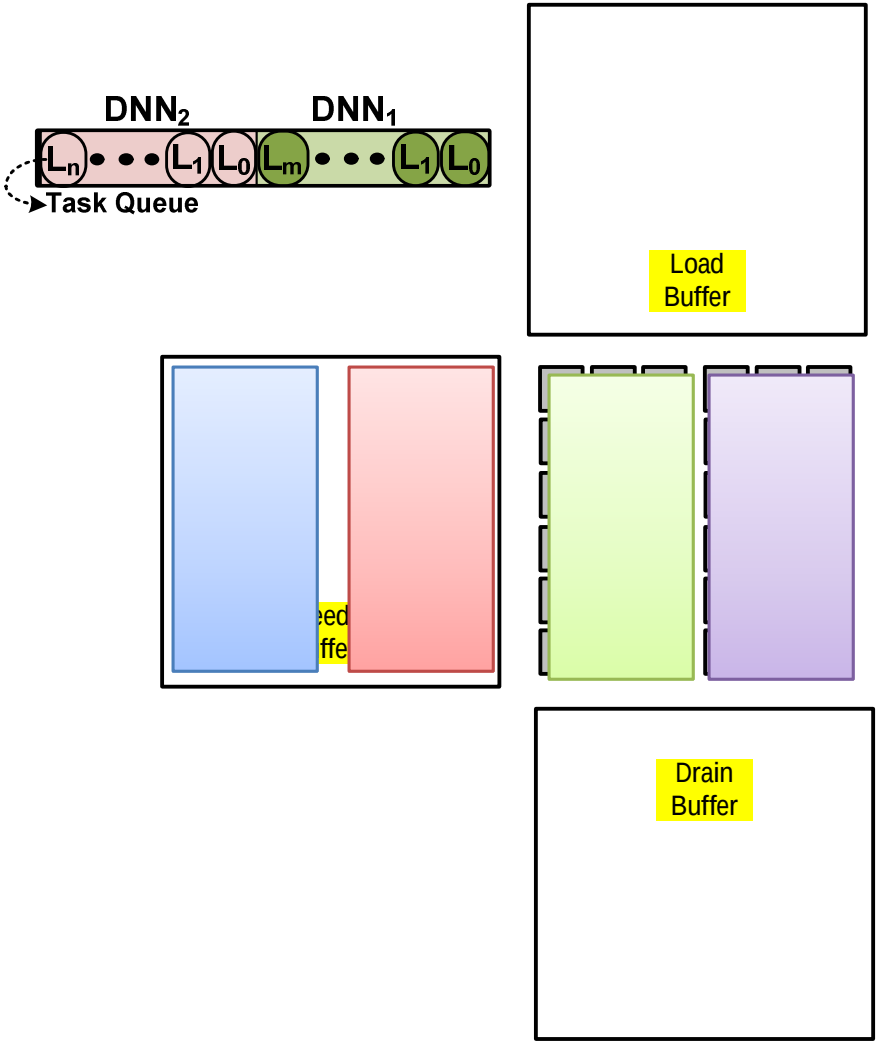
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x*,*y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
    Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
      //step ❷
      Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
        Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
          //step ❶
          Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
            Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

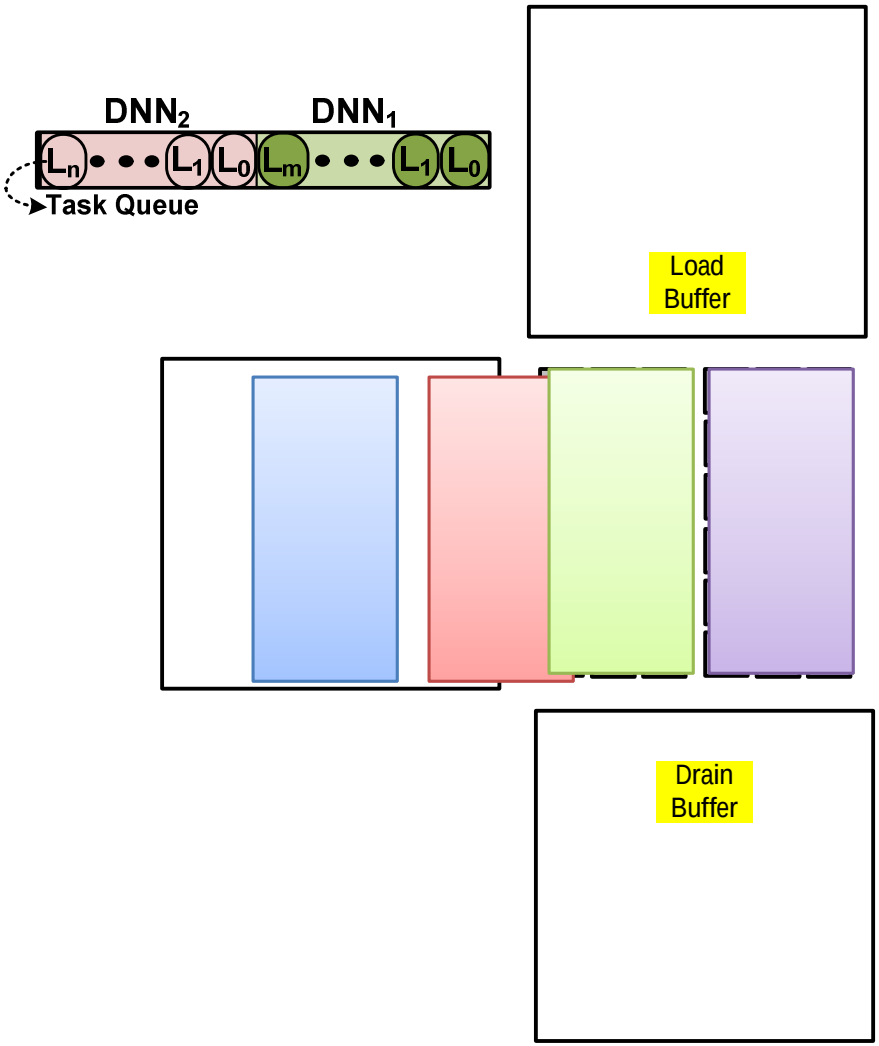
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x*,*y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x\prime} = PE_x$ //Partition size in x dimension
  $PE_{y\prime} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

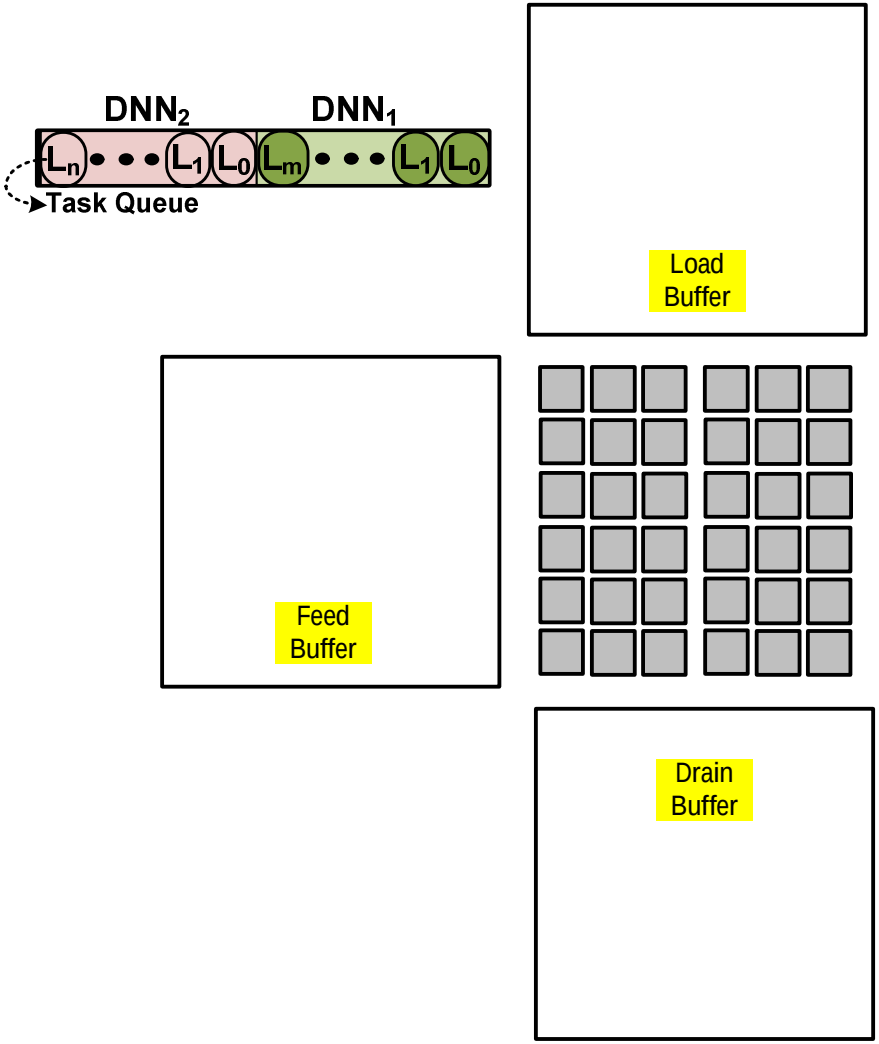
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x*,*y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

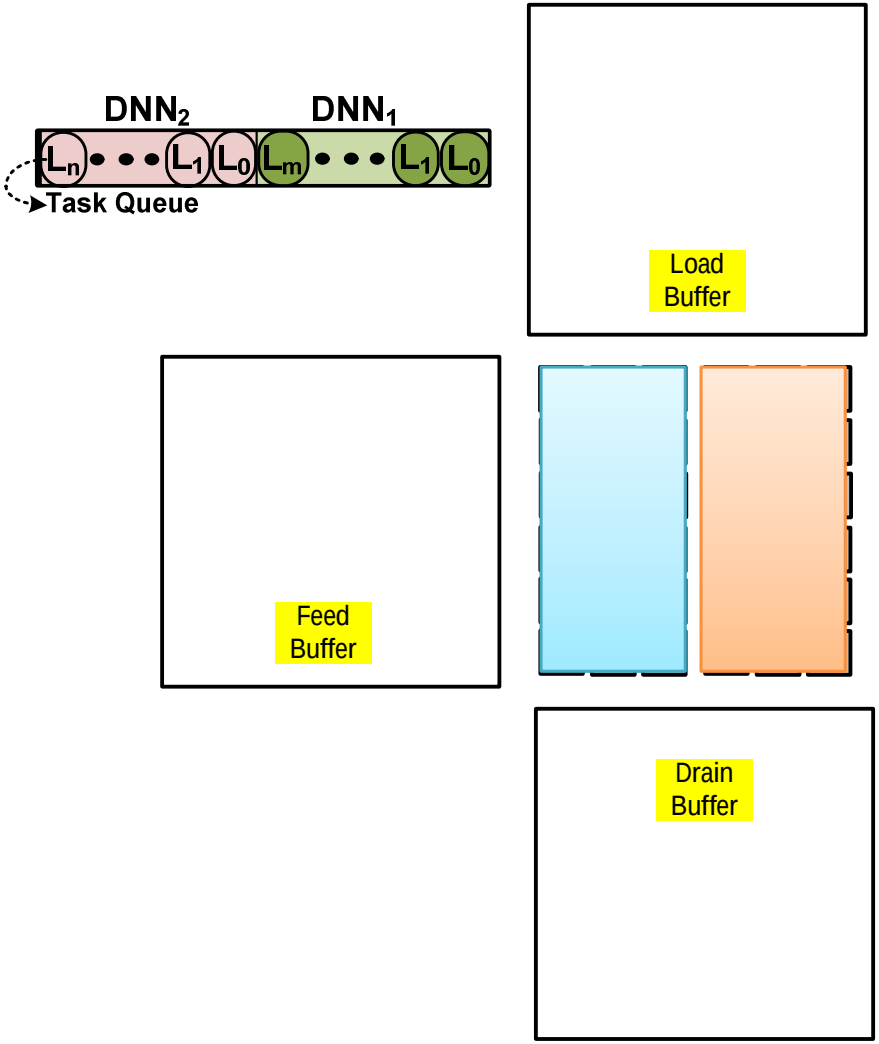
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x,y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



Dynamic Partitioning Algorithm

Algorithm1: Dynamic Resource Partitioning

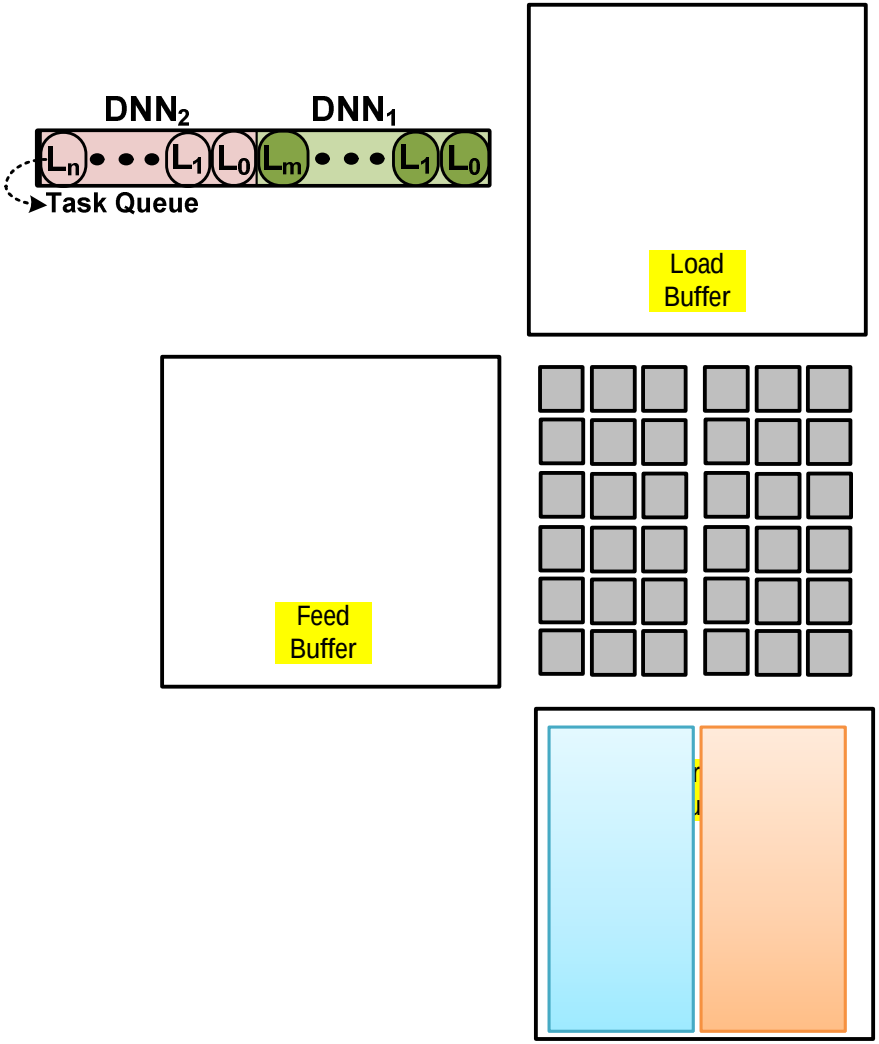
Inputs: DNNG (m, n, A_t, E_t)
SA architecture (Storage(*size*), PE(*x*,*y*))
Outputs: Partition size estimation (PE(x′,y′))
Task assignment($\tau_i \rightarrow PE(x', y')$)

Queue [*index*] ← ∅
if (*index*==1) //First DNNG inside queue
  Assign τ_0 to PE(x,y) //Assign to all PEs with no partitioning
else
  for All l_i with $A_{ti} \leq E_{t1}$
    Number of partitions= Number of DNNGs inside Queue
    Call Partition_Calculation (*Number of available layers*)
    Call Task_Assignment (*Available Layers*)
    Call Partioned_Weight_Stationery (*Available Layers*)
  end_for
end_if

Function Partition_Calculation (*Number of available layers*)
  $PE_{x'} = PE_x$ //Partition size in x dimension
  $PE_{y'} = \left\lfloor \frac{PE_y}{Number\ of\ Available\ layes} \right\rfloor$ //Partition size in y dimension
  return (PE(x′,y′))
end_function

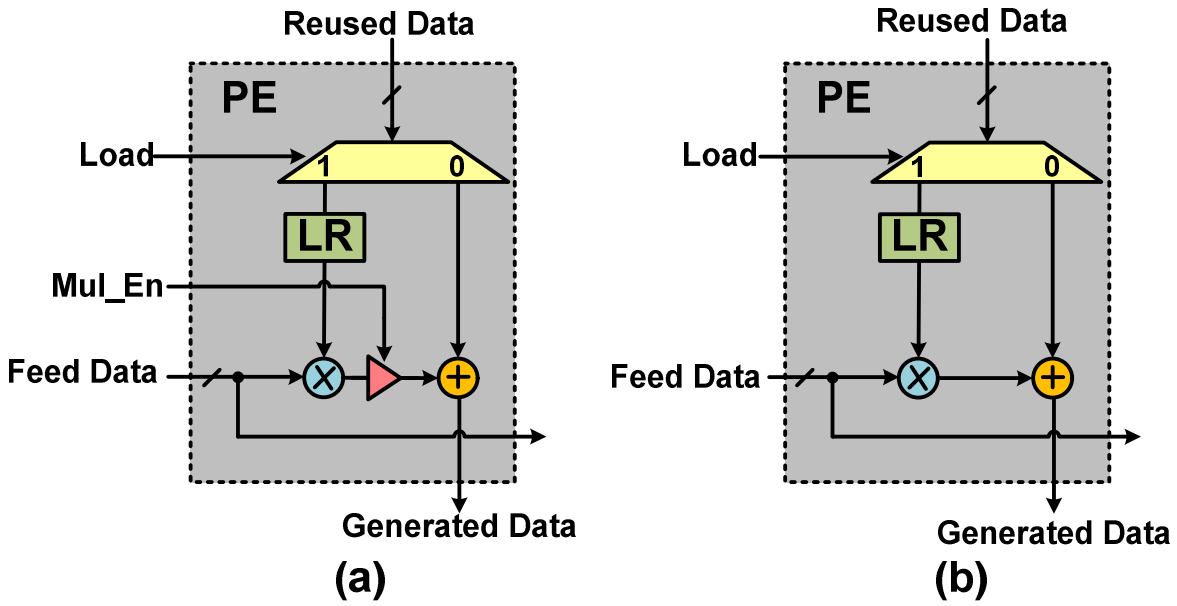
Function Task_Assignment (*Available layers*)
  for All available l_i
    Number of operations of the layer l_i, $Opr(l_i) = \prod shapes$
    Sort $Opr(l_i)$ from high to low
    Assign l_i with the highest Opr to available partition
  end_for
  return (Assigned l_i)
end_function

Function Partitioned_Weight_Stationery (*Available layers*)
  n = Number of available layers
  for All partitions n
    //step ❸
    Temporal_for PE [partition column$_Y$] to Drain Buffer [partition column]
      Parallel_for PE [partition row$_X$] to Drain Buffer [partition row]
        //step ❷
        Temporal_for Feed Buffer [partition column] on PE [partition column$_Y$]
          Parallel_for Feed Buffer [partition row] on PE [partition row$_X$]
            //step ❶
            Parallel_for Load Buffer [partition row] to PE [partition row$_X$]
              Parallel_for Load Buffer [partition column] to PE [partition column$_Y$]
    end_parallel_temporal_for
  end_for
end_function



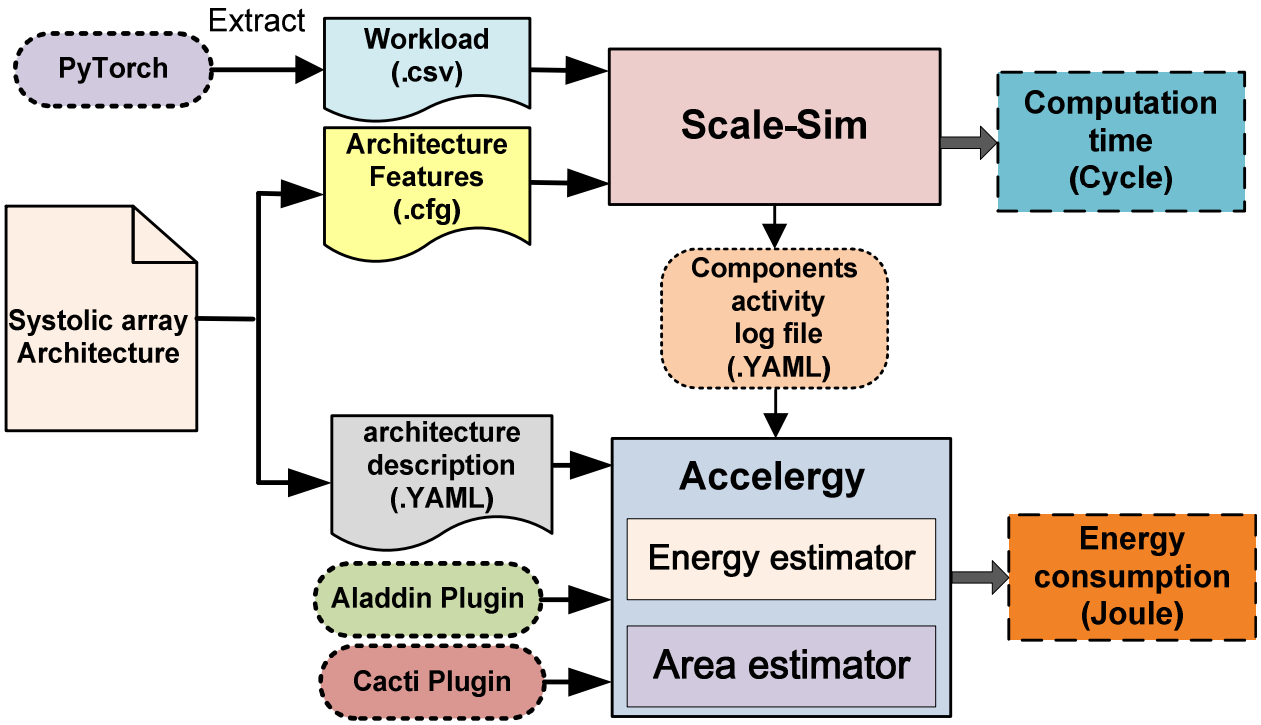
The proposed PE

- Systolic-array Based DNN Accelerator



The Simulation Toolchain

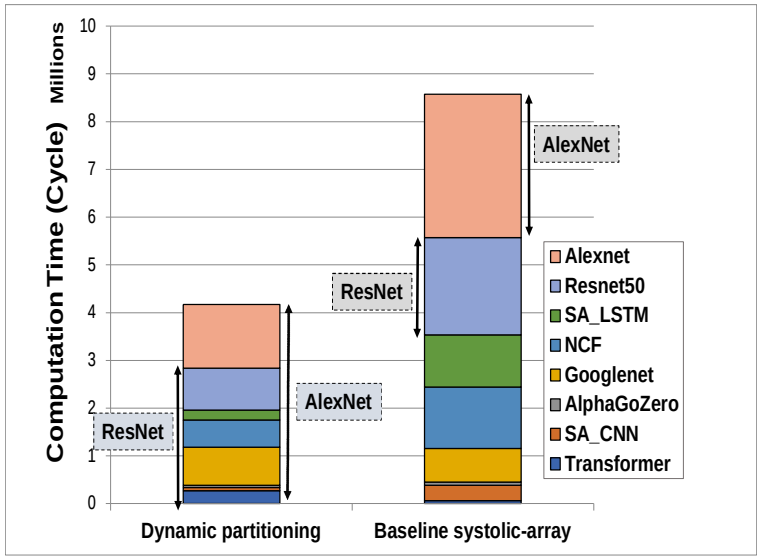
- Systolic-array Based DNN Accelerator

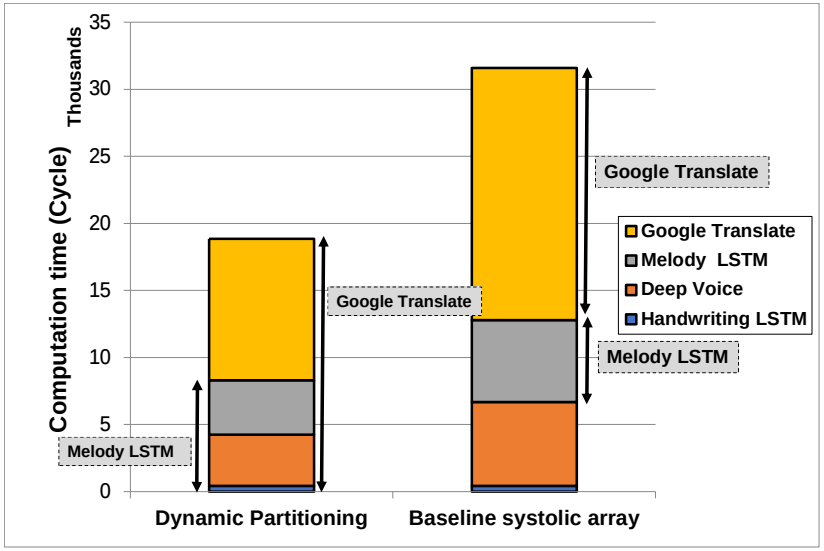


The Simulation Workloads

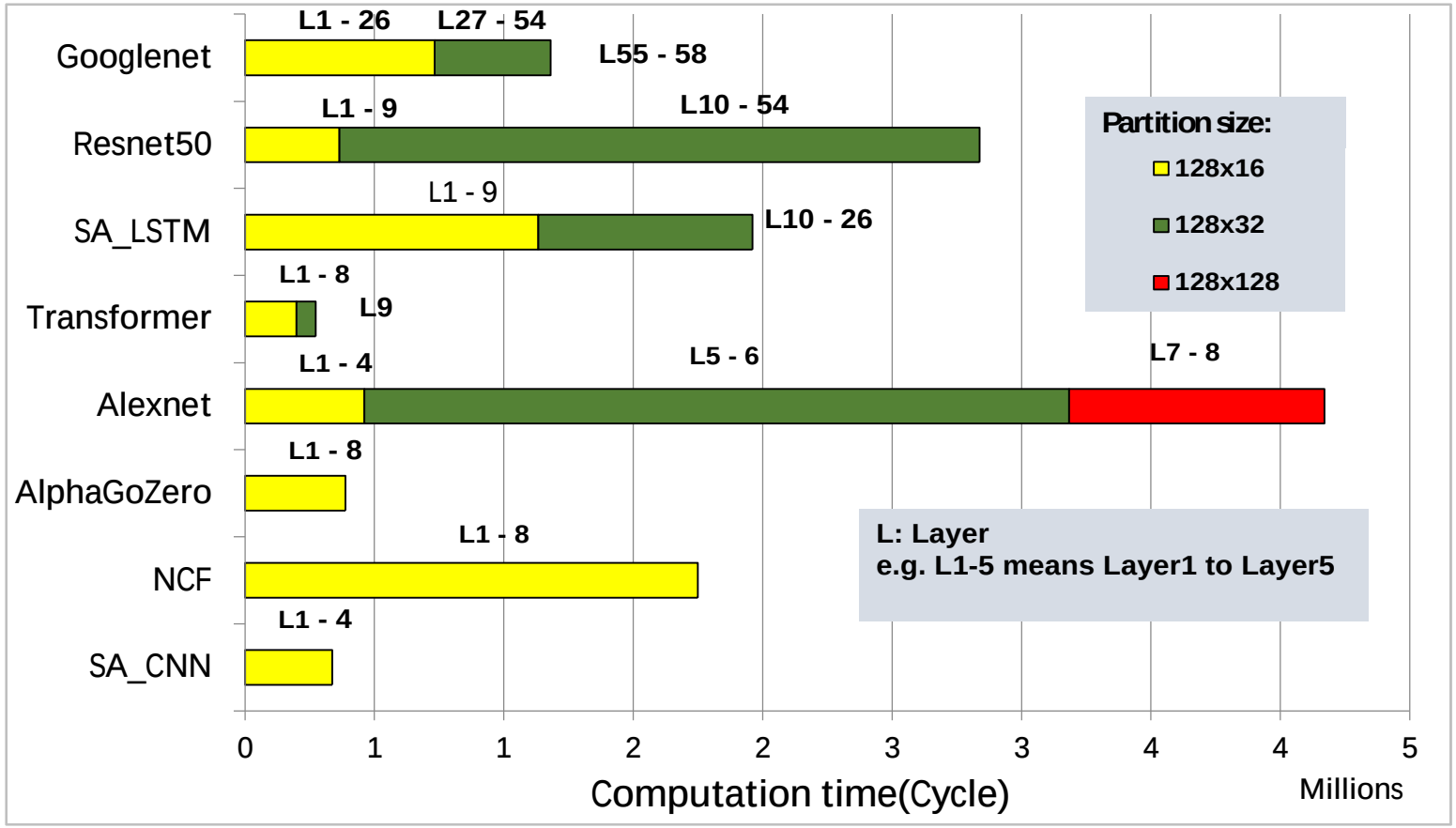
Type	Domain	Training model
Multi-domain / Heavy load workload	Image classification	AlexNet[16]
		ResNet50[17]
		GoogleNet[18]
	Sentiment analysis	Sentiment analysis CNN (SA_CNN)[19]
		Sentiment analysis LSTM (SA_LSTM)[20]
	Recommendation system	Neural collaborative filter (NCF)[21]
	Intelligent search	AlphGoZero[22]
	Natural language processing	Transformer [23]
Recurrent Neural Network / Lightload workload	Melody extraction	Melody LSTM [24]
	Language translation	Google Translate [25]
	Text to speech	Deep voice[26]
	Handwriting recognition	Handwriting LSTM [27]

The Simulation Results

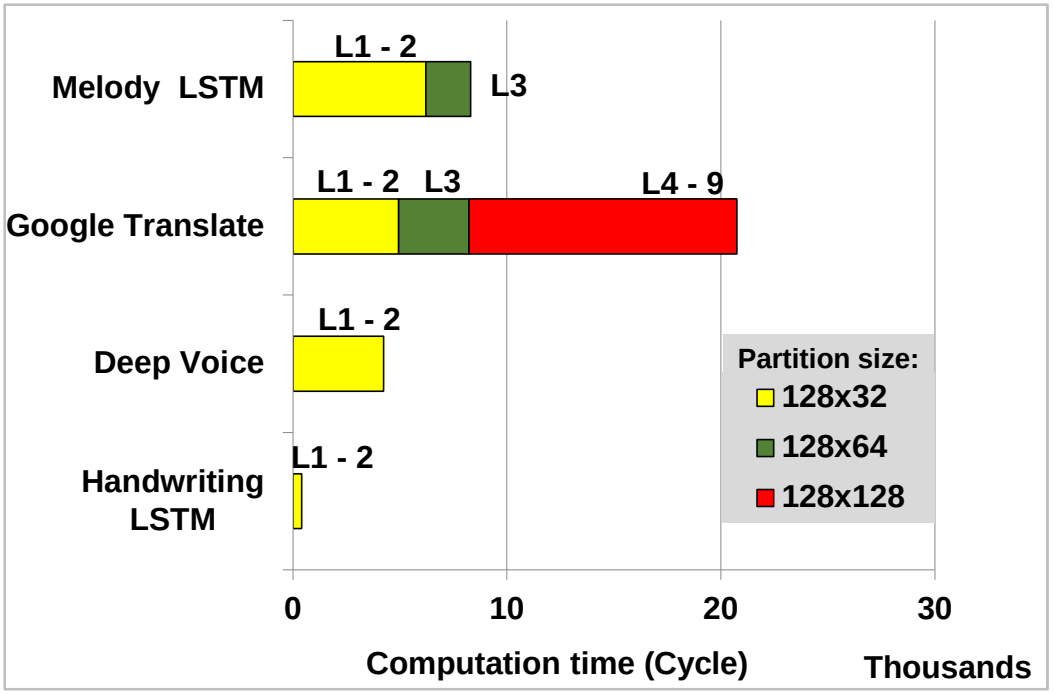




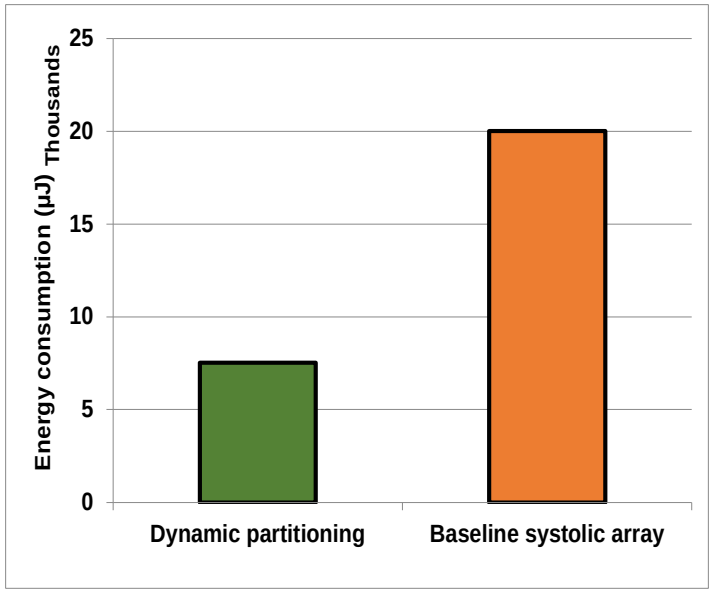
The Simulation workloads

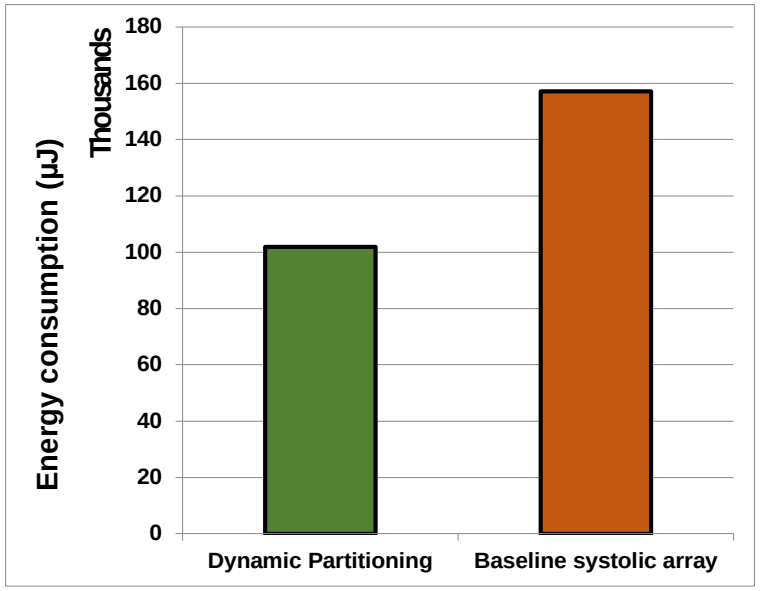


The Simulation workloads



The Simulation workloads





Conclusion

- Concurrent running multiple DNNs
- Sharing accelerator’s storage and processing resources
- Dynamic partitioning algorithm:
  - Increasing resource utilization
  - Reducing computation time
- Partitioned weight stationary dataflow
- Minor logic modification of systolic array

Thank you for your attention