Optimizing CNN Inference on Multicore Scratchpad Architectures in Real Time Systems

Chiara Daini[1,2]

[1]University of Lille, France
[2]Inria, Sycomores Team, France

November 29, 2024

Supervisors:

- **Giuseppe Lipari**: Professor at the University of Lille, head of the SYCOMORES team
- **Houssam Zahaf**: Associate Professor at Nantes University

SYCOMORES Team:

- A joint team between Inria and the University of Lille
- Focus: Design and analysis of real-time embedded systems
- Domains: Safety-critical systems (avionics, automotive, IoT), real-time scheduling, synchronous languages, static code analysis and proof assistants

Optimizing CNN Inference on Multicore Scratchpad Architectures in Real Time Systems

Optimizing CNN Inference on Multicore Scratchpad Architectures in Real Time Systems

Optimizing CNN Inference on Multicore Scratchpad Architectures in Real Time Systems

Optimizing CNN Inference on Multicore Scratchpad Architectures in Real Time Systems

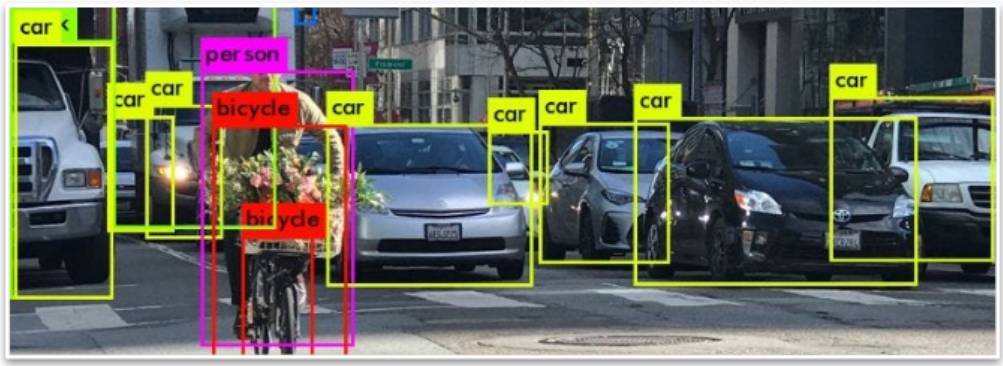
Timeline

- Motivations
- Convolution Neural Networks
- Multicore Scratchpad Architectures
- Real Time Systems
- Optimization Model
- Experiments
- Conclusion

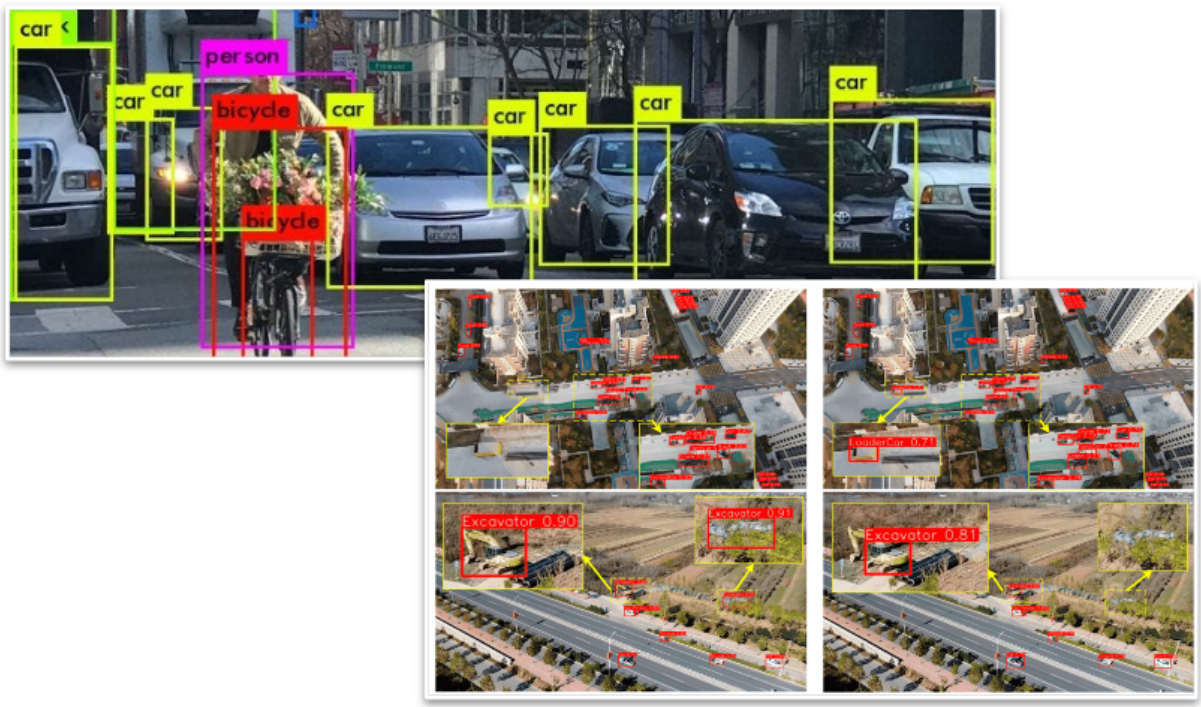
Timeline

- **Motivations**
- Convolution Neural Networks
- Multicore Scratchpad Architectures
- Real Time Systems
- Optimization Model
- Experiments
- Conclusion

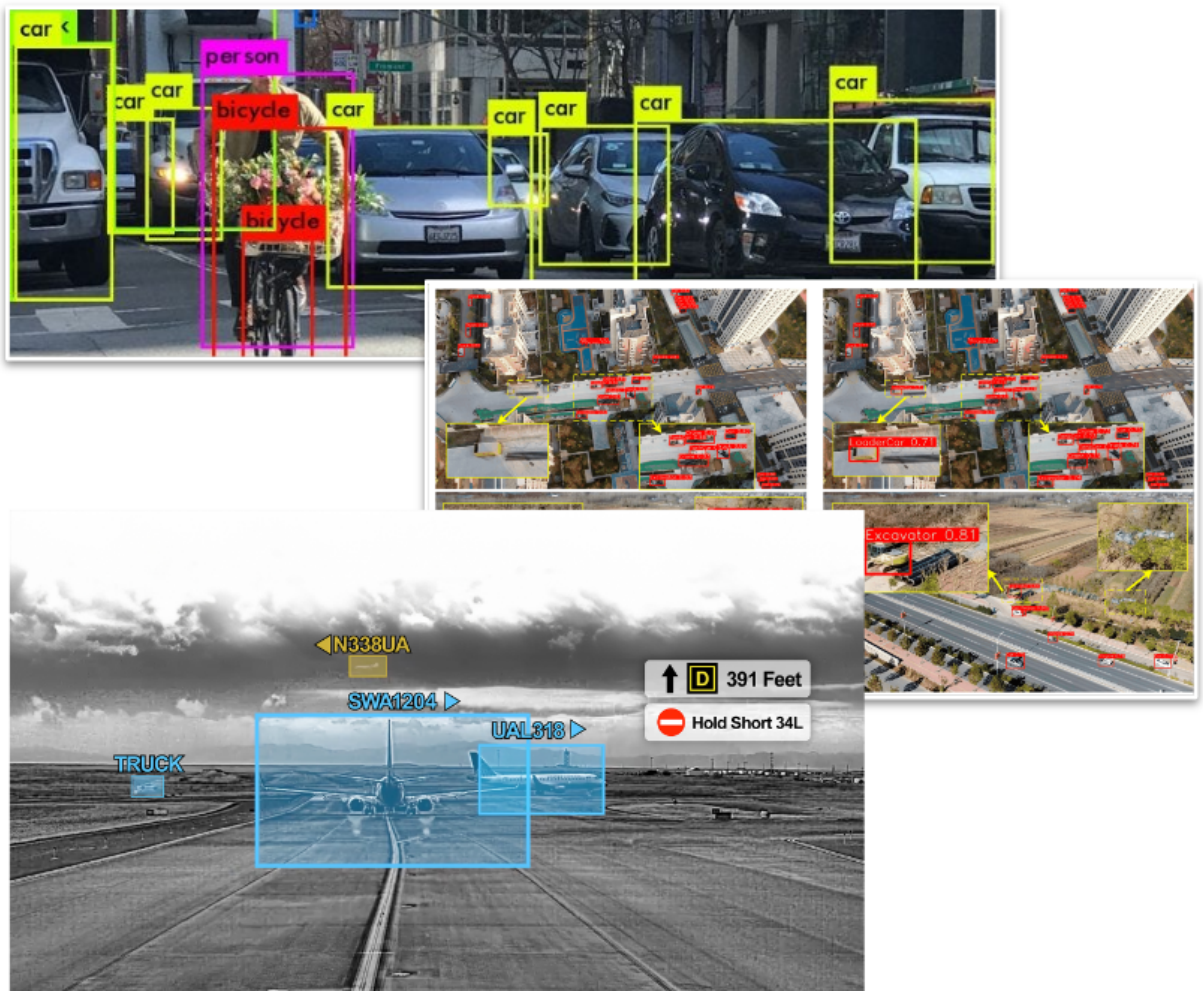
Motivations



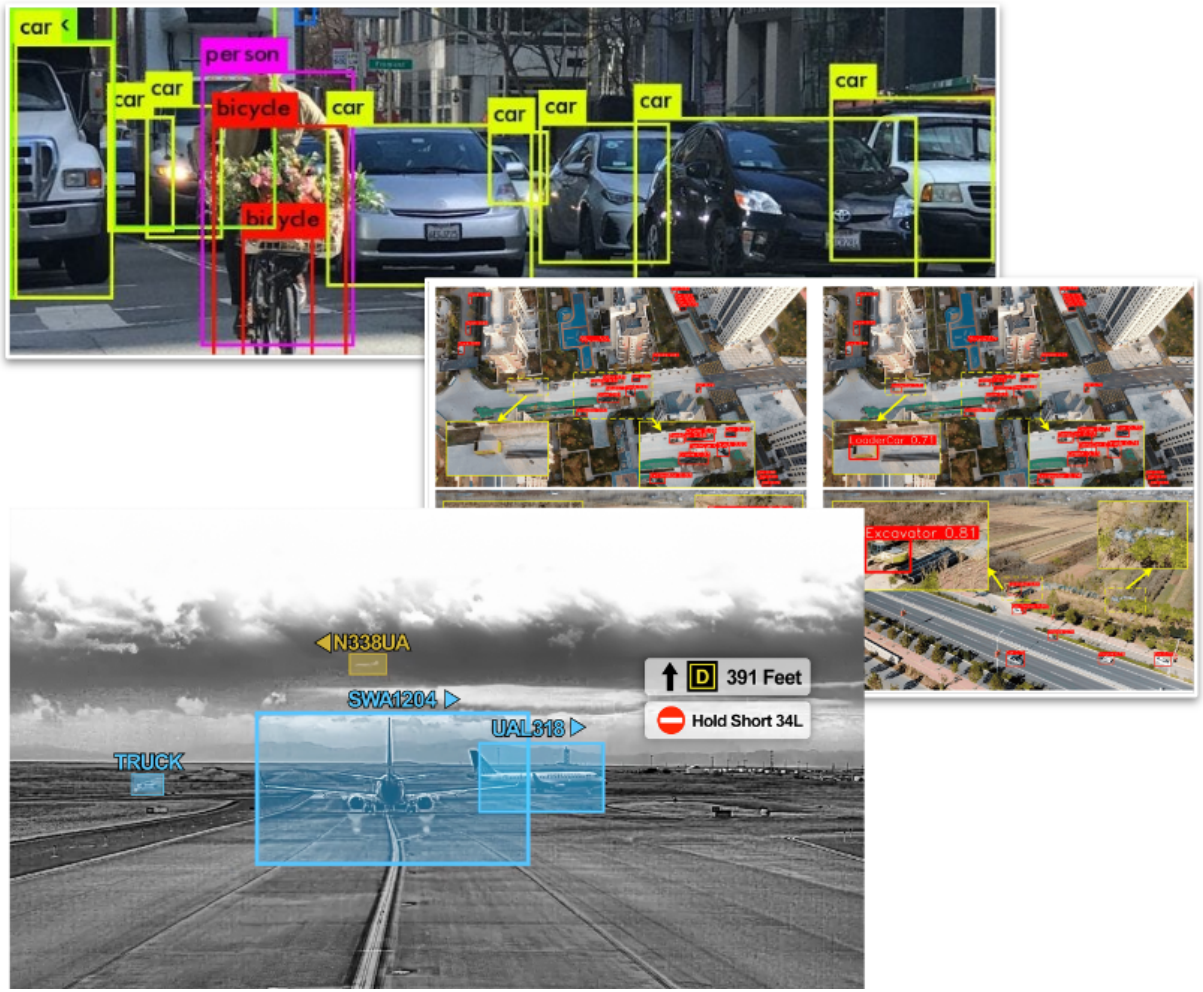
Motivations



Motivations



Motivations



Modern AI applications, particularly **CNNs**, are increasingly computationally intensive, requiring efficient execution to meet **performance** and **real-time constraints.**

Multicore scratchpad-based systems provide **high predictability** but introduce challenges such as memory contention and task scheduling, impacting overall efficiency and predictability.

Timeline

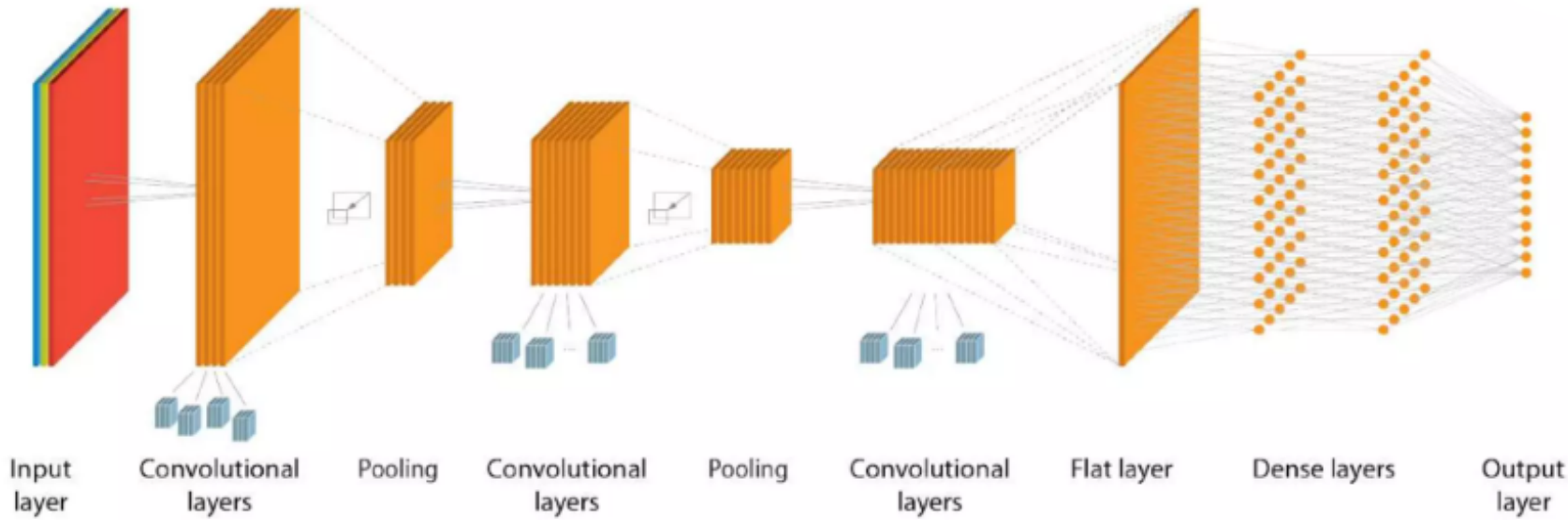
- Motivations
- **Convolution Neural Networks**
- Multicore Scratchpad Architectures
- Real Time Systems
- Optimization Model
- Experiments
- Conclusion

Convolution Neural Networks

CNNs are specifically designed for processing **matrices of pixels**.
They are used in:

- Image classification
- Image recognition

The core operation in a CNN is the **convolution**.



Convolution Neural Networks

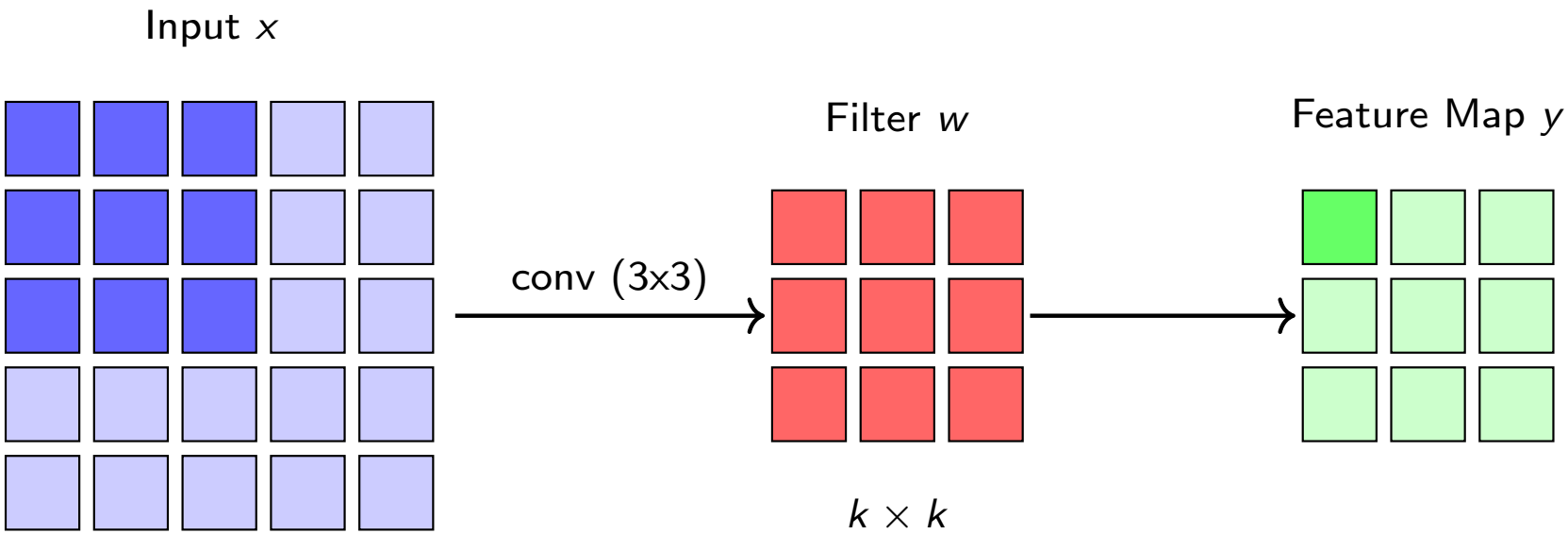
Training phase: process where the CNN learn patterns from data already classified
Inference phase: application of a trained model to new, unseen data to make classifications

Convolution Neural Networks

Training phase: process where the CNN learn patterns from data already classified

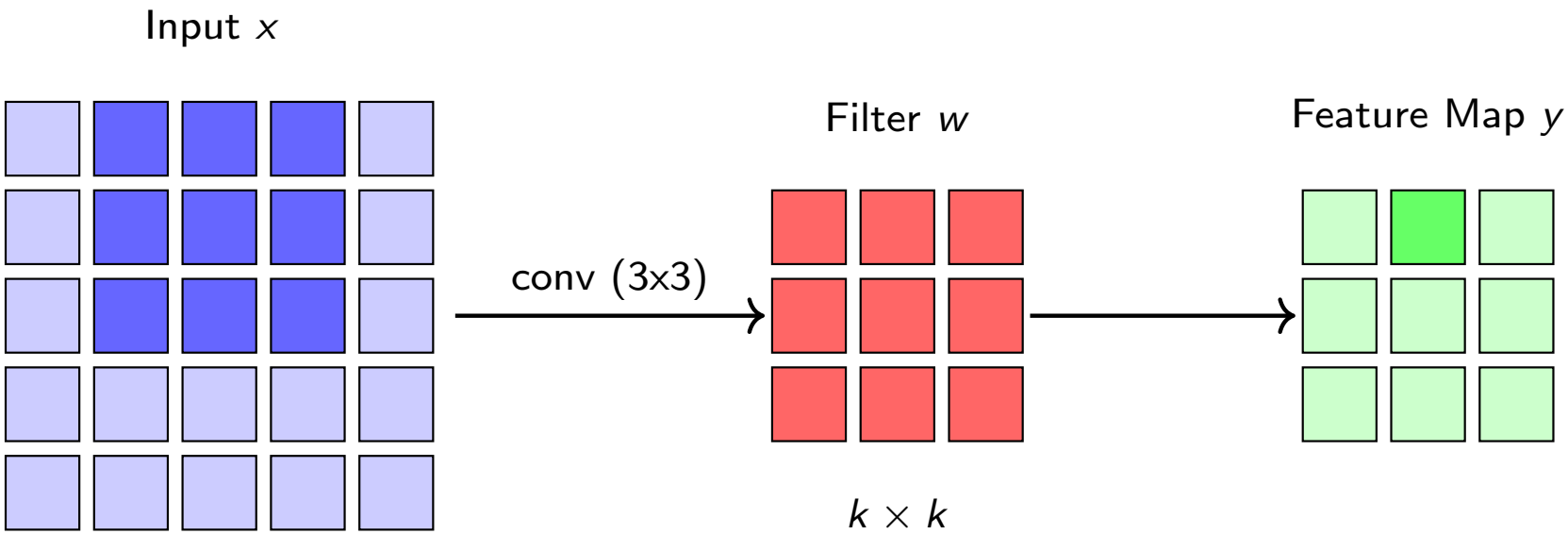
Inference phase: application of a trained model to new, unseen data to make classifications

Convolution Operation



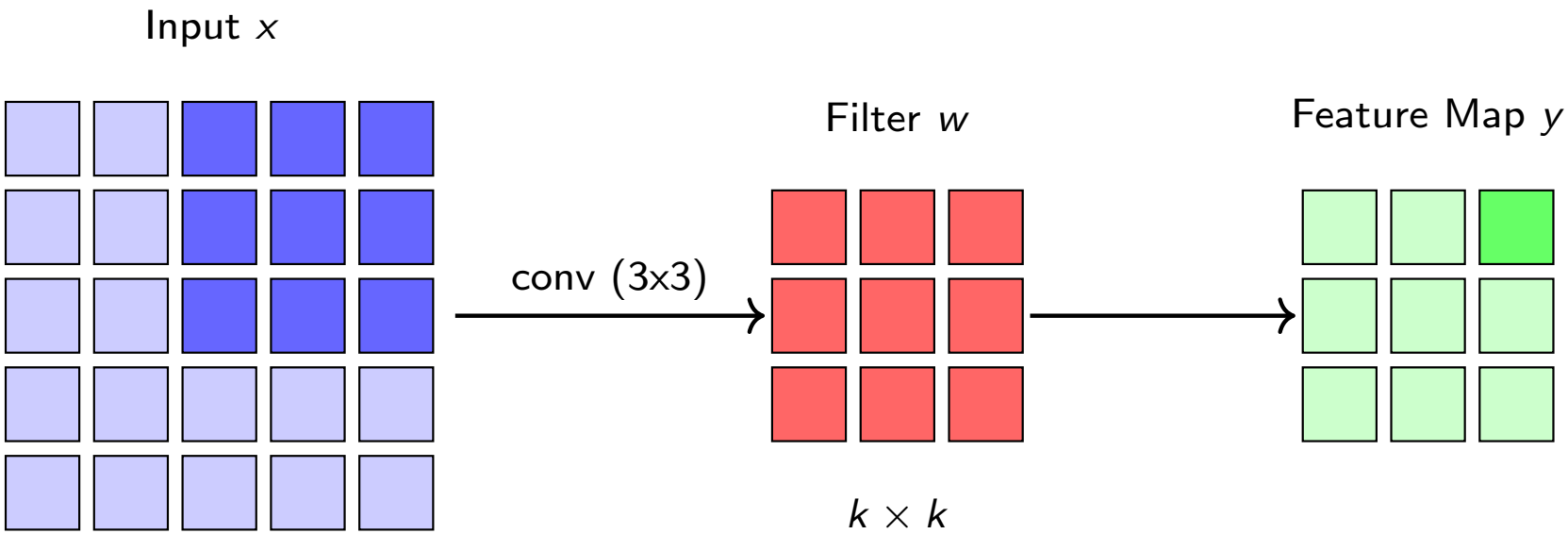
$$\sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w_{m,n} \cdot x_{i+m,j+n} = y_{i,j}$$

Convolution Operation



$$\sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w_{m,n} \cdot x_{i+m,j+n} = y_{i,j}$$

Convolution Operation



$$\sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w_{m,n} \cdot x_{i+m,j+n} = y_{i,j}$$

Convolution Operation

Edge Detection Filter: $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Timeline

- Motivations
- Convolution Neural Networks
- **Multicore Scratchpad Architectures**
- Real Time Systems
- Optimization Model
- Experiments
- Conclusion

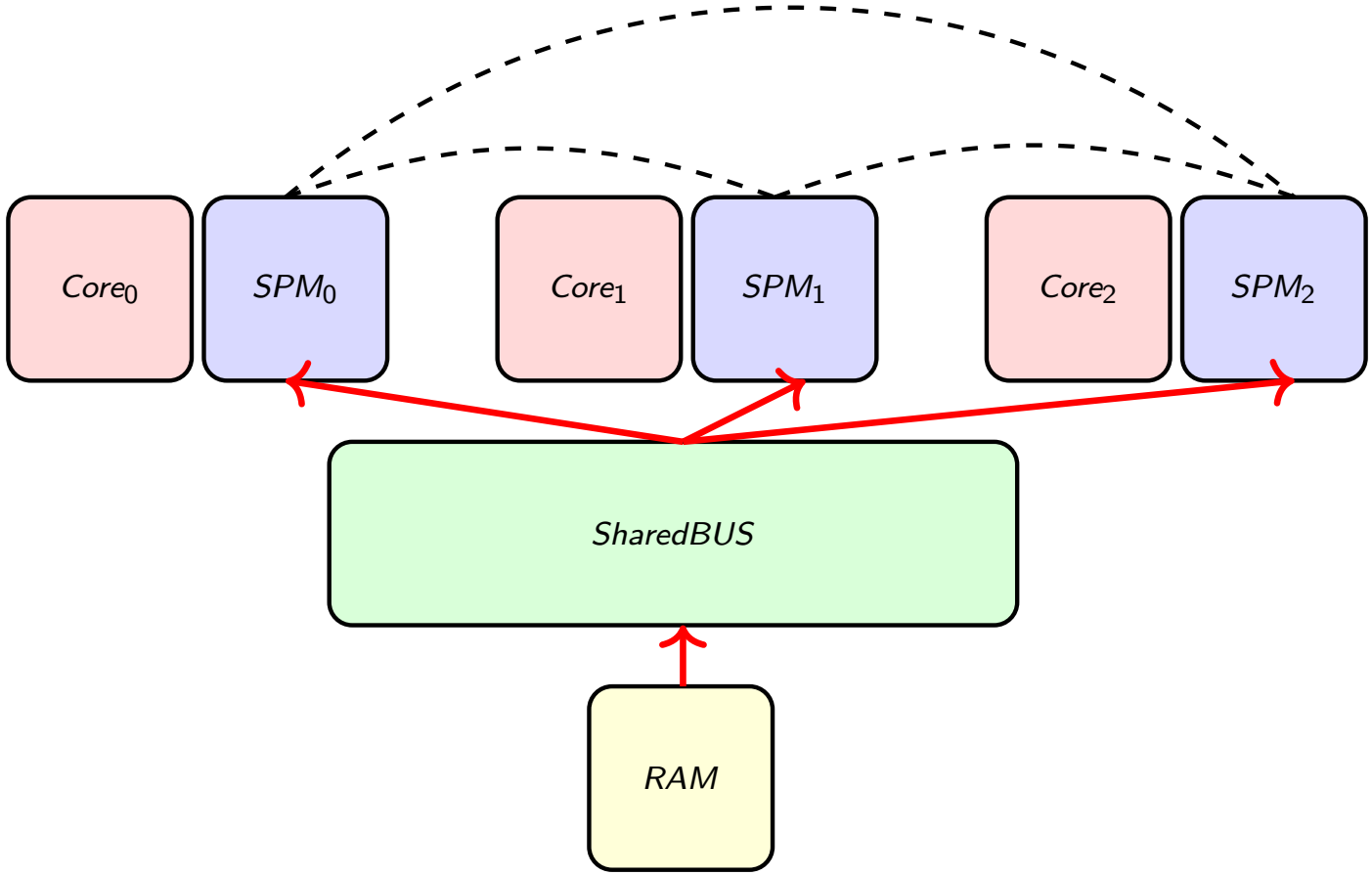
Multicore Scratchpad Architectures

- Multicore architectures utilize **multiple processing units** (*cores*) to perform parallel computations
- Each core is equipped with a local **Scratchpad Memory (SPM)**, a small, fast, software-managed memory used to store frequently accessed data
- The **Shared Bus** performs data transfer between scratchpads and RAM
- Common in real-time and embedded systems due to predictable memory access patterns

Multicore Scratchpad Architectures

Feature	**SPM**	**Cache**
Management	Manual (software)	Automatic (hardware)
Predictability	High, deterministic	Low, dynamic
Performance	Deterministic, software-tuned	High if few misses
Use Cases	Real-time, embedded	General-purpose

Multicore Scratchpad Architectures



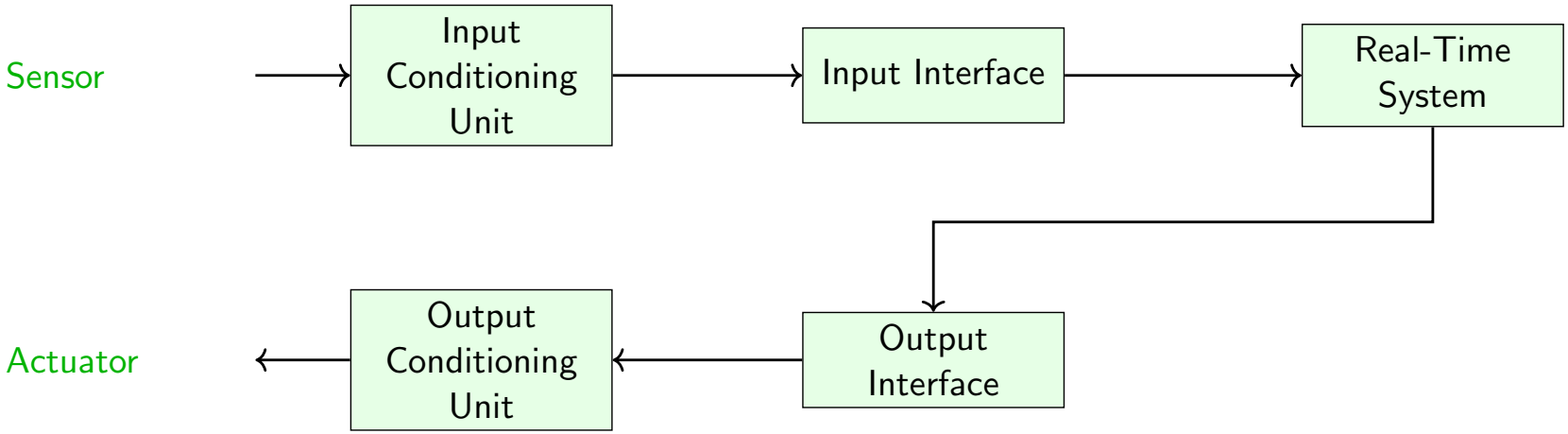
→ = Memory Transfer (RAM to SPMs) via Bus

- - - = Data Transfer (Inter-SPM Communication)

Timeline

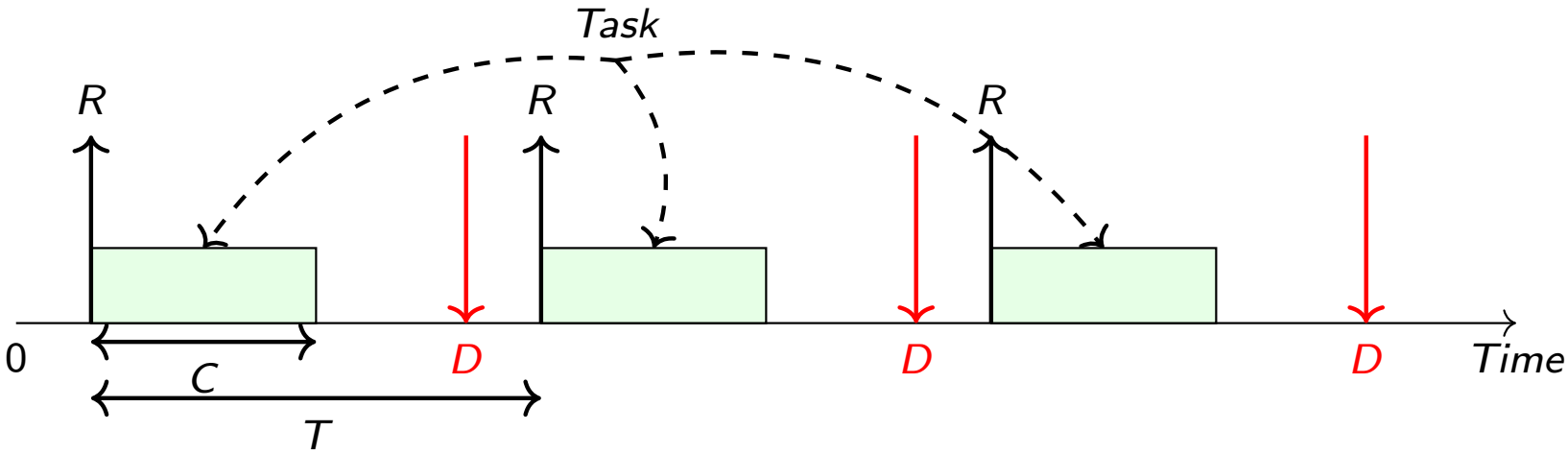
- Motivations
- Convolution Neural Networks
- Multicore Scratchpad Architectures
- **Real Time Systems**
- Optimization Model
- Experiments
- Conclusion

Real-Time Systems



- A *real-time system* is a computing system designed to perform tasks within strict timing constraints.
- Processes input data (e.g., from sensors) and generates responses (e.g., to actuators) in a predictable and timely manner.
- Correctness of the system depends not only on the logical results of computations but also on their timing.

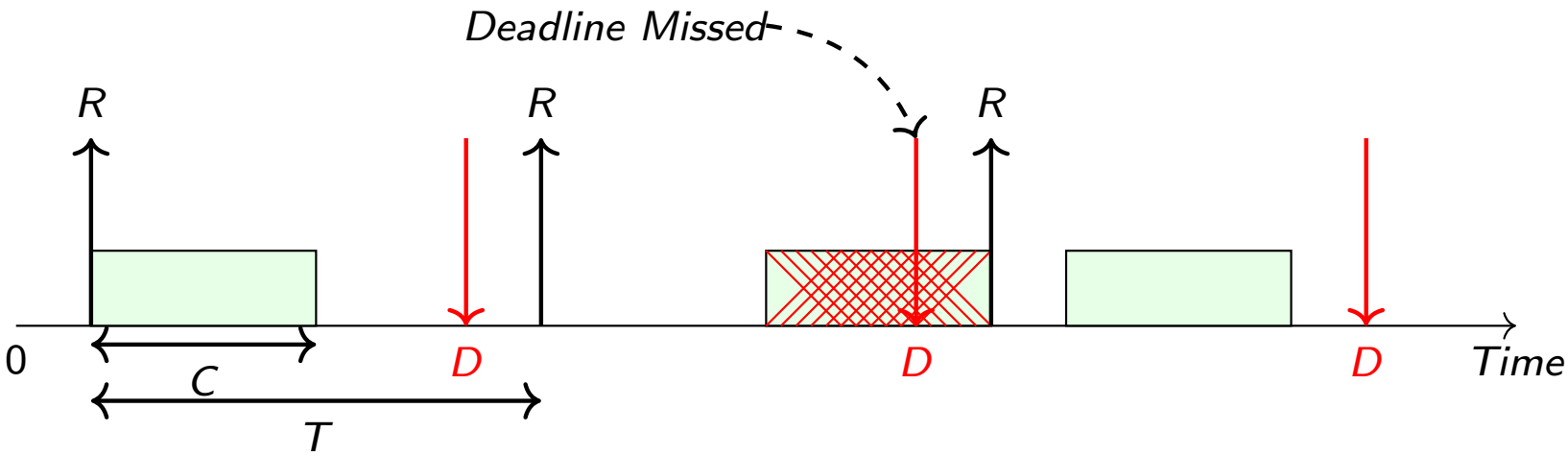
Real-Time Systems



Where:

- **R**: Release Time
- **C**: Execution time
- **D**: Deadline
- **T**: Period

Real-Time Systems



In a **hard real-time system**, missing a deadline means the task was not completed on time, which can cause system failure or unsafe behavior.

Timeline

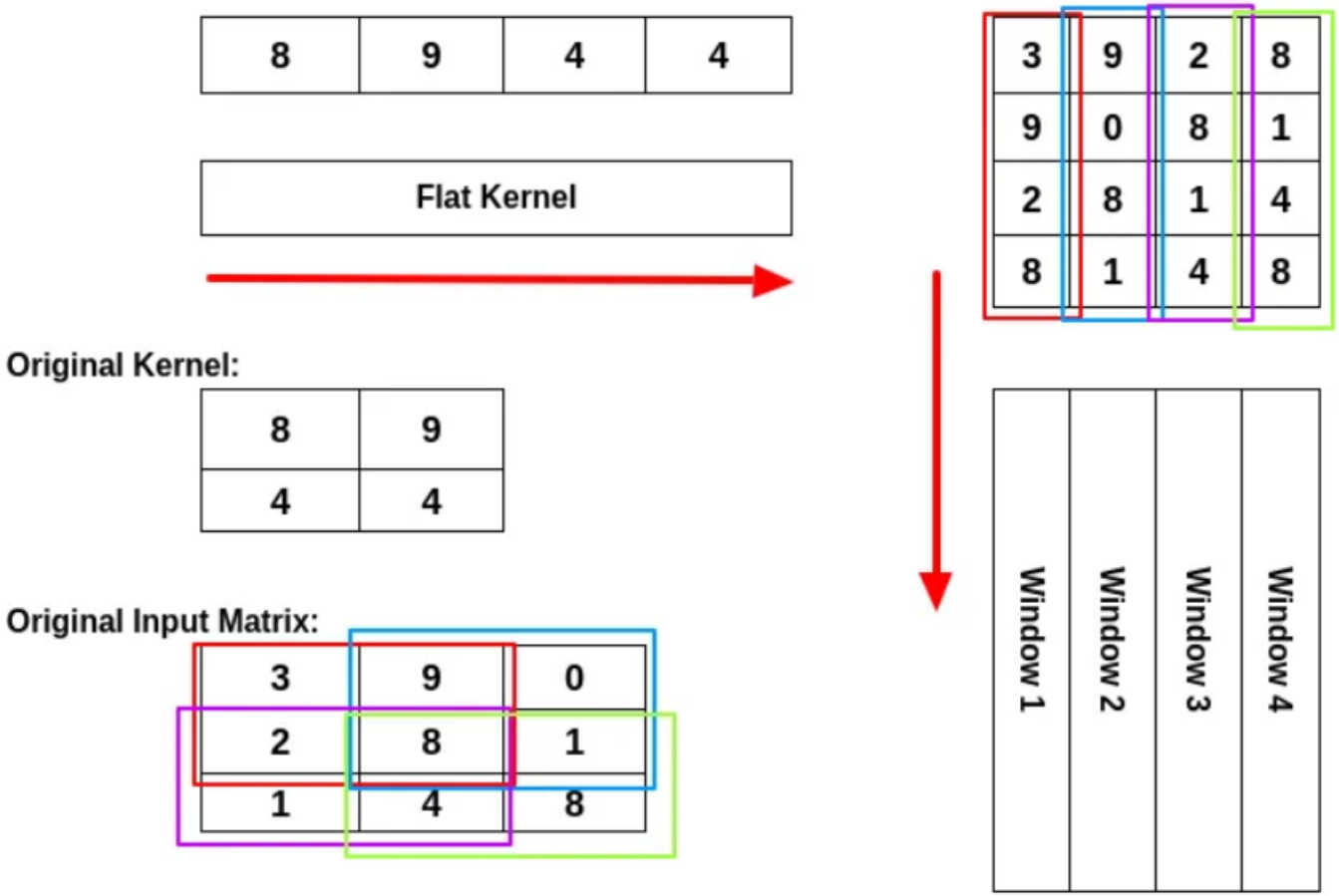
- Motivations
- Convolution Neural Networks
- Multicore Scratchpad Architectures
- Real Time Systems
- **Optimization Model**
- Experiments
- Conclusion

Traditional Techniques for Optimizing Convolution

GEMM (General Matrix Multiplication):

- Converts convolution into matrix multiplication using *im2col*
- Leverages optimized BLAS libraries for efficiency.
- **Overhead:**
  - Large memory usage due to im2col transformation (data duplication)
  - Additional preprocessing to reshape inputs and filters

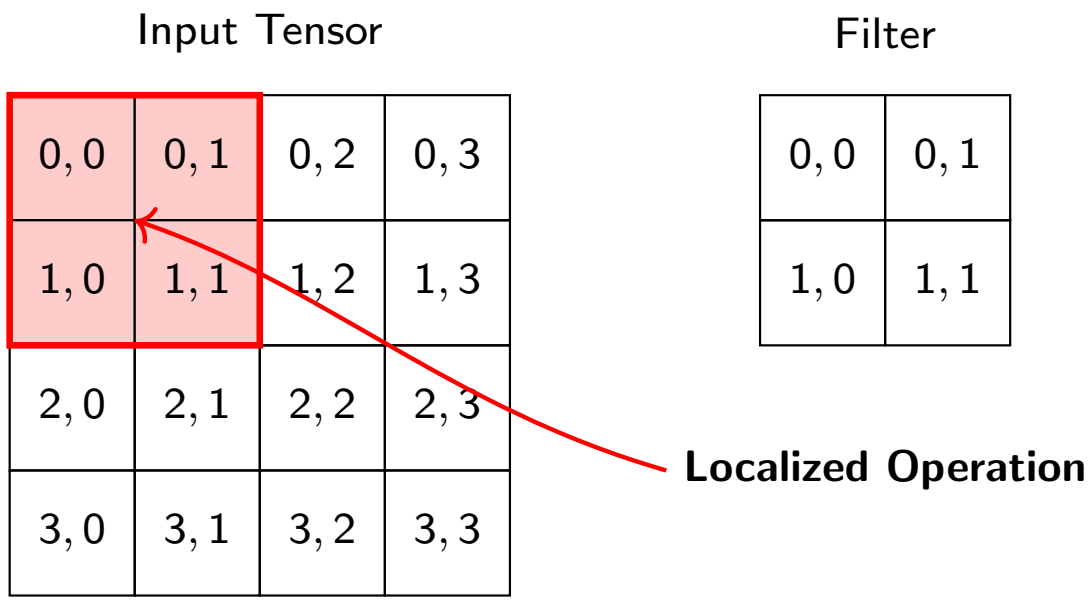
Traditional Techniques for Optimizing Convolution



System Model

Localized operation (v): a set of arithmetic operations to be performed and the set of data to be loaded before starting the computation.

- C(v): Computation time for arithmetic operations.
- M(v): Data required to be loaded into local memory before computation.



System Model

The *localized operations* are executed on a platform with the following characteristics:

- P: Number of processing cores.
- M: Scratchpad memory size available per core.

Additionally, the CNN must meet the specified deadline, D.

System Model

Mapping function: gives for a localized operation v the core p where v is allocated:

$$\text{map}(v) = p$$

Operation Merge: A merge of a set of localized operations $\mathcal{M}$, e.g. $\mathcal{M} = \{v_1, v_2\}$, is a new localized operation v' such that:

- $\forall (v_1, v_2) \in \mathcal{M}^2 : \text{map}(v_1) = \text{map}(v_2) = \text{map}(v')$

and the localized operation v' is defined as:

- $\text{C}(v') = \sum_{v \in \mathcal{M}} \text{C}(v)$
- $\text{M}(v') = \bigcup_{v \in \mathcal{M}, \text{pred}(v) \notin \mathcal{M}} \text{M}(v)$

Example merge

Input tensor

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Filter

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

Example merge

v_0

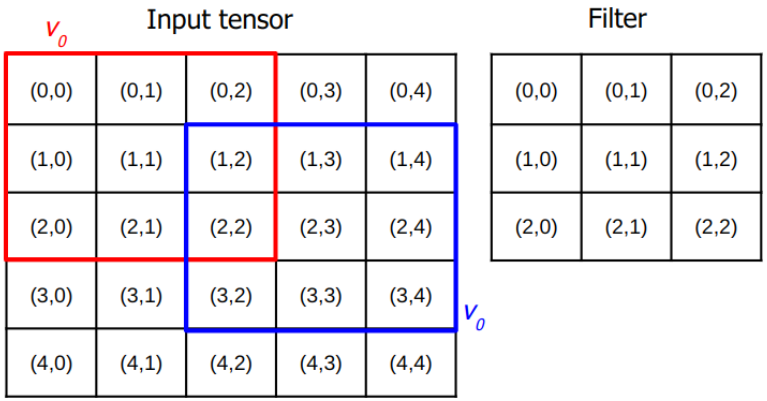
Input tensor

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

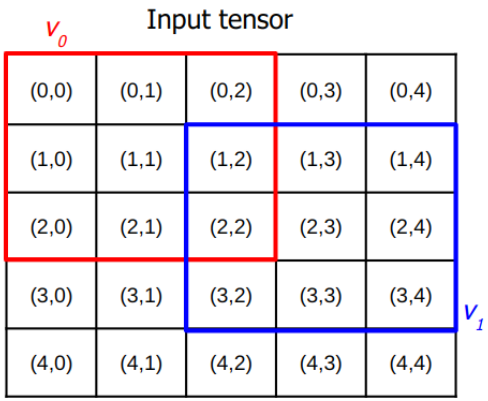
Filter

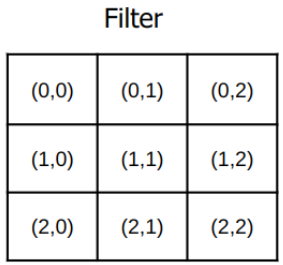
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

Example merge



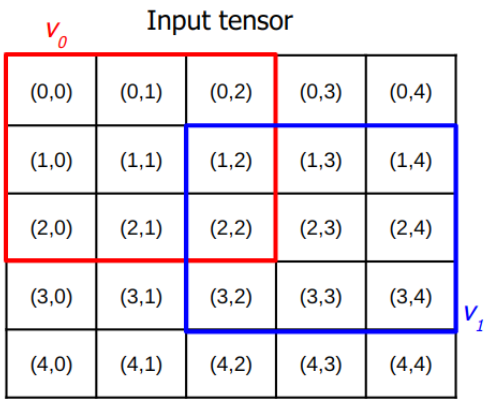
Example merge





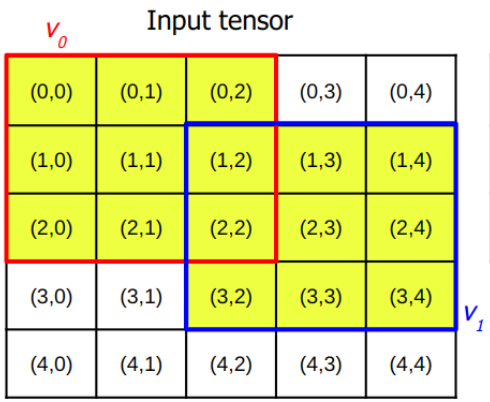
$\text{map}(v_0) = \text{map}(v_1) = p$

Example merge



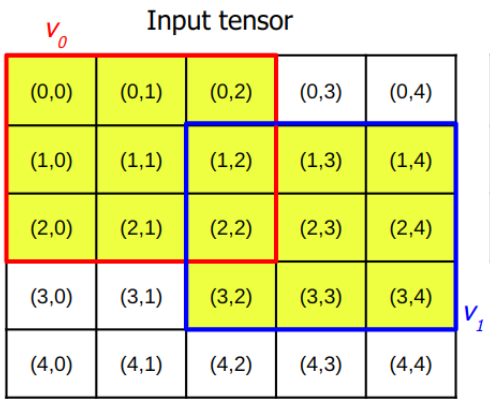
$\text{map}(v_0) = \text{map}(v_1) = p \Longrightarrow \text{map}(v_2) = p$

Example merge



$\text{map}(v_0) = \text{map}(v_1) = p \Longrightarrow \text{map}(v_2) = p$

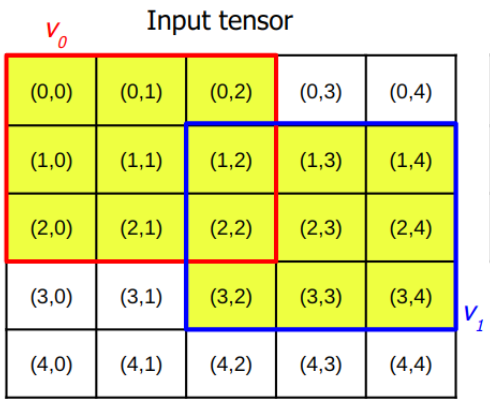
Example merge



$\text{map}(v_0) = \text{map}(v_1) = p \Longrightarrow \text{map}(v_2) = p$

$C(v_2) = C(v_0) + C(v_1)$

Example merge

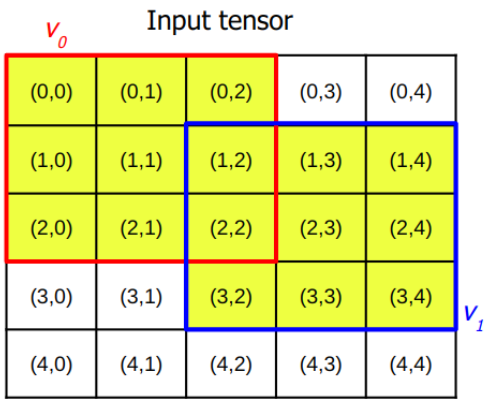


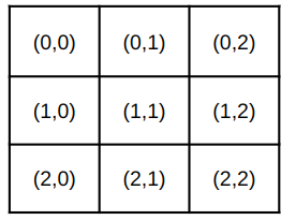
$map(v_0) = map(v_1) = p \Longrightarrow map(v_2) = p$

$C(v_2) = C(v_0) + C(v_1)$

$M(v_2) = U\ (\ M(v_0), M(v_1)\ )$

Example merge

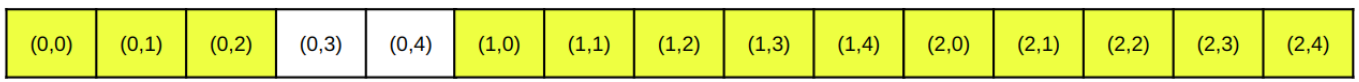


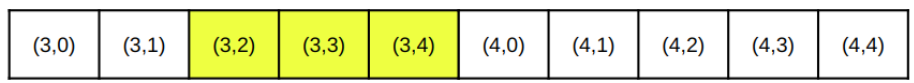


$map(v_0) = map(v_1) = p \Longrightarrow map(v_2) = p$

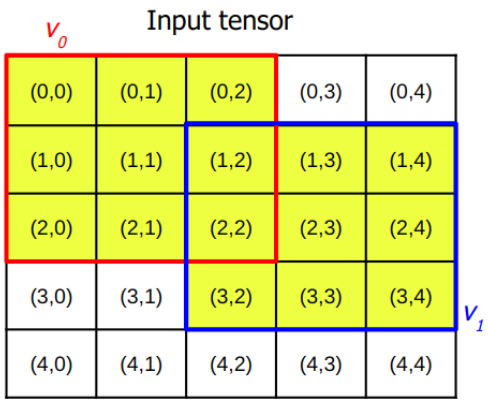
$C(v_2) = C(v_0) + C(v_1)$

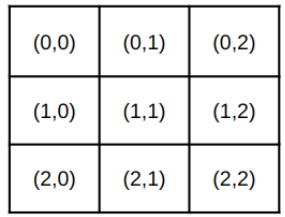
$M(v_2) = U\ (\ M(v_0), M(v_1)\ )$





Example merge

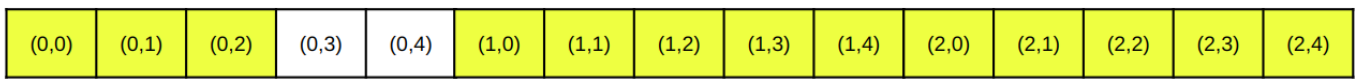


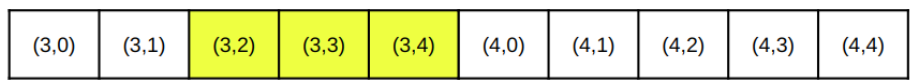


$map(v_0) = map(v_1) = p \Longrightarrow map(v_2) = p$

$C(v_2) = C(v_0) + C(v_1)$

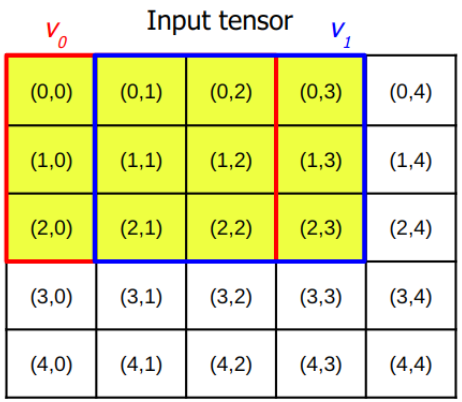
$M(v_2) = U\ (\ M(v_0), M(v_1)\ )$

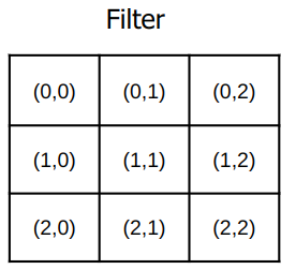




PIXELS COPIED = 16

Example merge

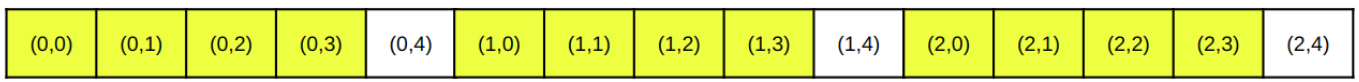


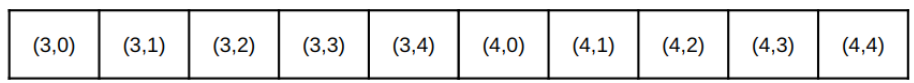


$map(v_0) = map(v_1) = p \Longrightarrow map(v_2) = p$

$C(v_2) = C(v_0) + C(v_1)$

$M(v_2) = U\ (\ M(v_0), M(v_1)\ )$





PIXELS COPIED = 12

Optimizer

The goal of the optimizer is to find the optimal mapping of the localized operations $\mathcal{V}$ to cores, such that:

- The mapping function $\text{map}(v)$ assigns each localized operation v to a core p, respecting the deadline D
- The memory constraint has to be satisfied, meaning that the total memory required for operations assigned to each core does not exceed the available scratchpad memory M.

Optimizer

Integer Linear Programming (ILP): optimization of a linear objective function under linear constraints where some or all variables are constrained to integer values

$$\begin{aligned} \text{Maximize:} \quad & c^T x, \\ \text{subject to:} \quad & Ax \leq b, \\ & x \in \mathbb{Z}^m. \end{aligned}$$

Optimizer

Operation Mapping Variable:

$$a_{i,l}^{p} = \begin{cases} 1 & \text{if } v_i^l \text{ is mapped to core } p, \\ 0 & \text{otherwise.} \end{cases}$$

Here, l represents the *layer* in the neural network, and v_i^l refers to the i-th localized operation within layer l.

Constraint:
Ensure that each operation is assigned to exactly one core:

$$\sum_{p \in [1..P]} a_{i,l}^{p} = 1$$

Optimizer

Memory Transfer Variables:

- $b_{i,l}(x, y)$: pixel required for operation v_i^l.
- $u_l^p(x, y)$: binary variable indicating if data is loaded on core p.

Constraint:

$$u_l^p(x, y) = \bigvee_{i \in [1..N_l]} a_{i,l}^p \cdot b_{i,l}(x, y)$$

Optimizer

v_0

Input tensor

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Filter

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

$\text{map}(v_0) = p$

core p:

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)

(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Optimizer

v_0

Input tensor

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Filter

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

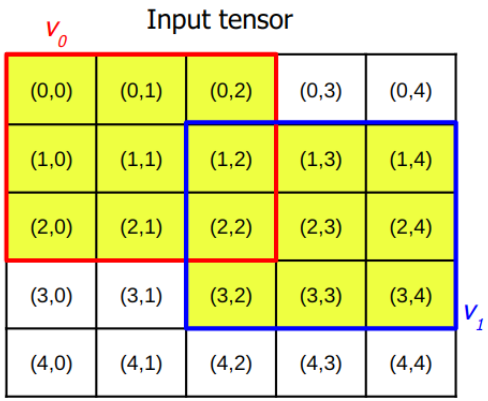
$\text{map}(v_0) = p$

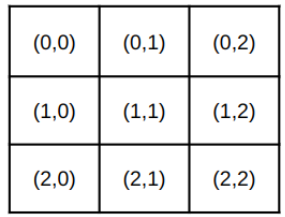
core p:

1	1	1	0	0	1	1	1	0	0	1	1	1	0	0

0	0	0	0	0	0	0	0	0	0

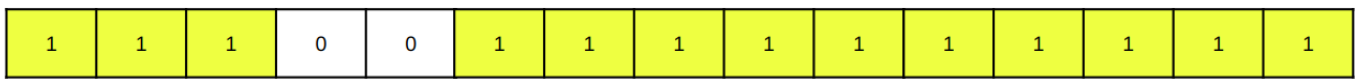
Optimizer





$\text{map}(v_0) = \text{map}(v_1) = p \Longrightarrow \text{map}(v_2) = p$

core p:



Optimizer

Execution Time:
The total execution time is divided into two components: the time required to transfer data to the scratchpad memory (SPM) and the time needed to perform computations. It is expressed as:

$$\mathbf{t}_p^l = \mathbf{t}_p^l(\text{mem}) + \mathbf{t}_p^l(\text{op})$$

Memory Constraint:
The total memory usage on each core must not exceed the available scratchpad memory M:

$$\sum_{x,y} u_l^p(x, y) \leq M$$

Distributed ILP

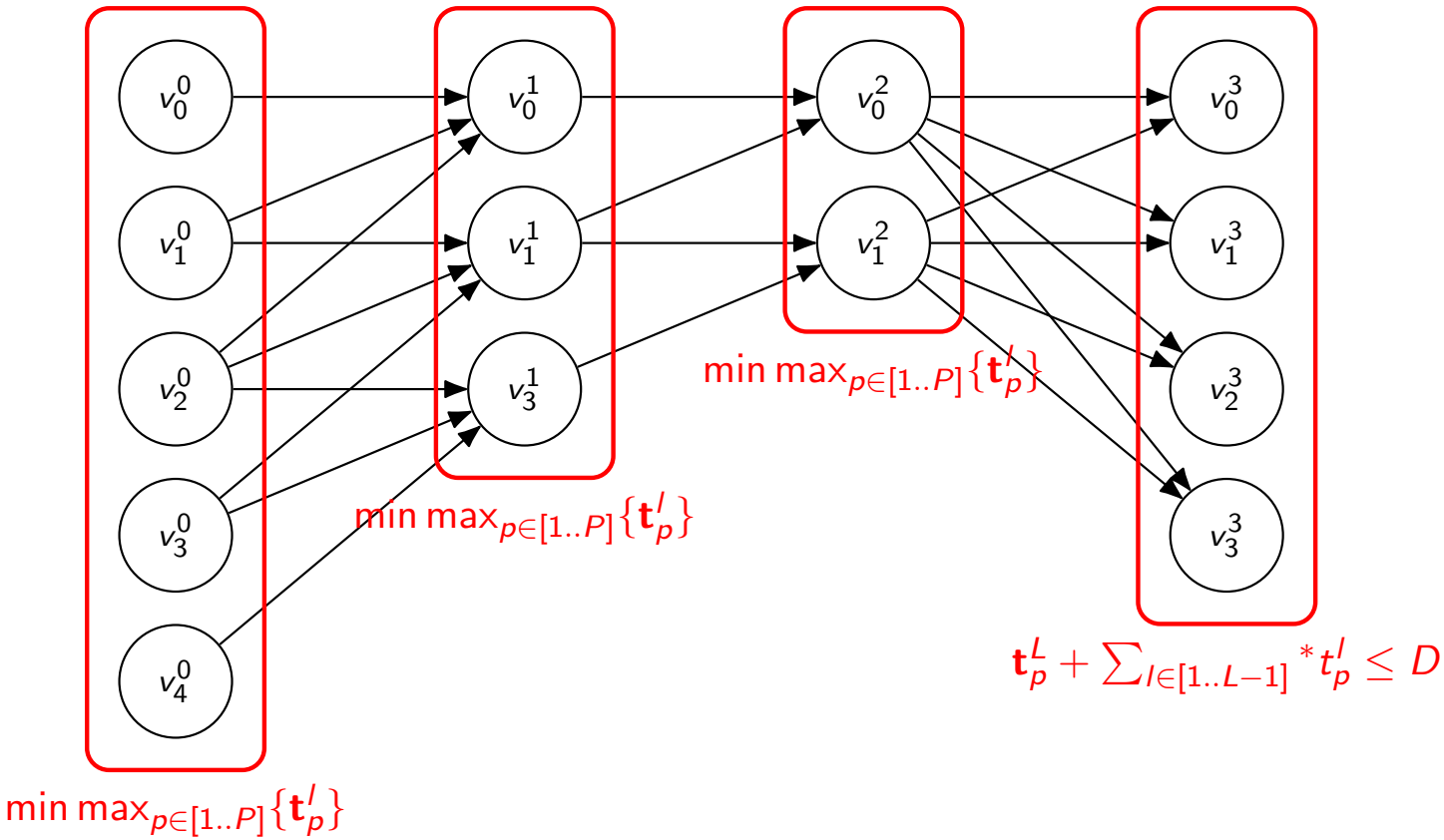
Decouple layers and solve independently, distributing the computational load across cores to minimize execution time. The objective for each hidden layer is:

$$^{*}t_p^l = \min \max_{p \in [1..P]} \{\mathbf{t}_p^l\}$$

For the last layer L, the constraint is:

$$\mathbf{t}_p^L + \sum_{l \in [1..L-1]} {}^{*}t_p^l \leq D$$

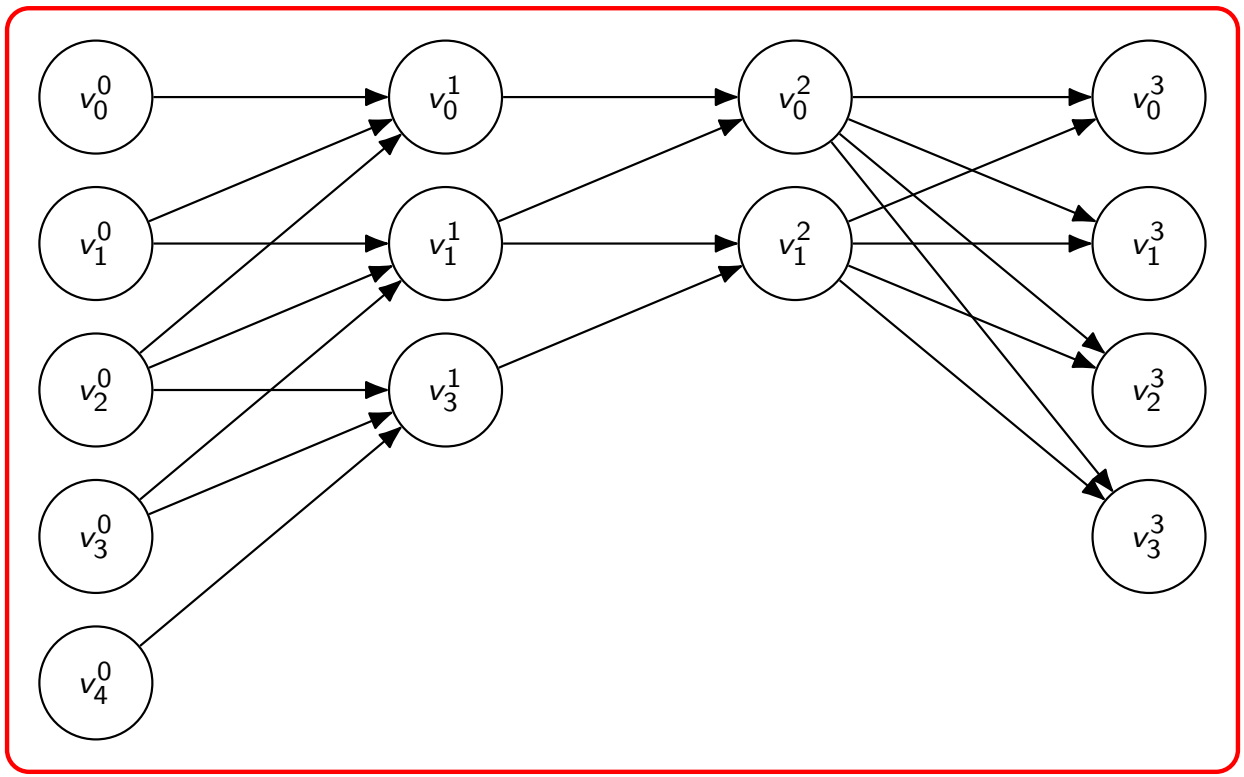
Distributed ILP



Global ILP

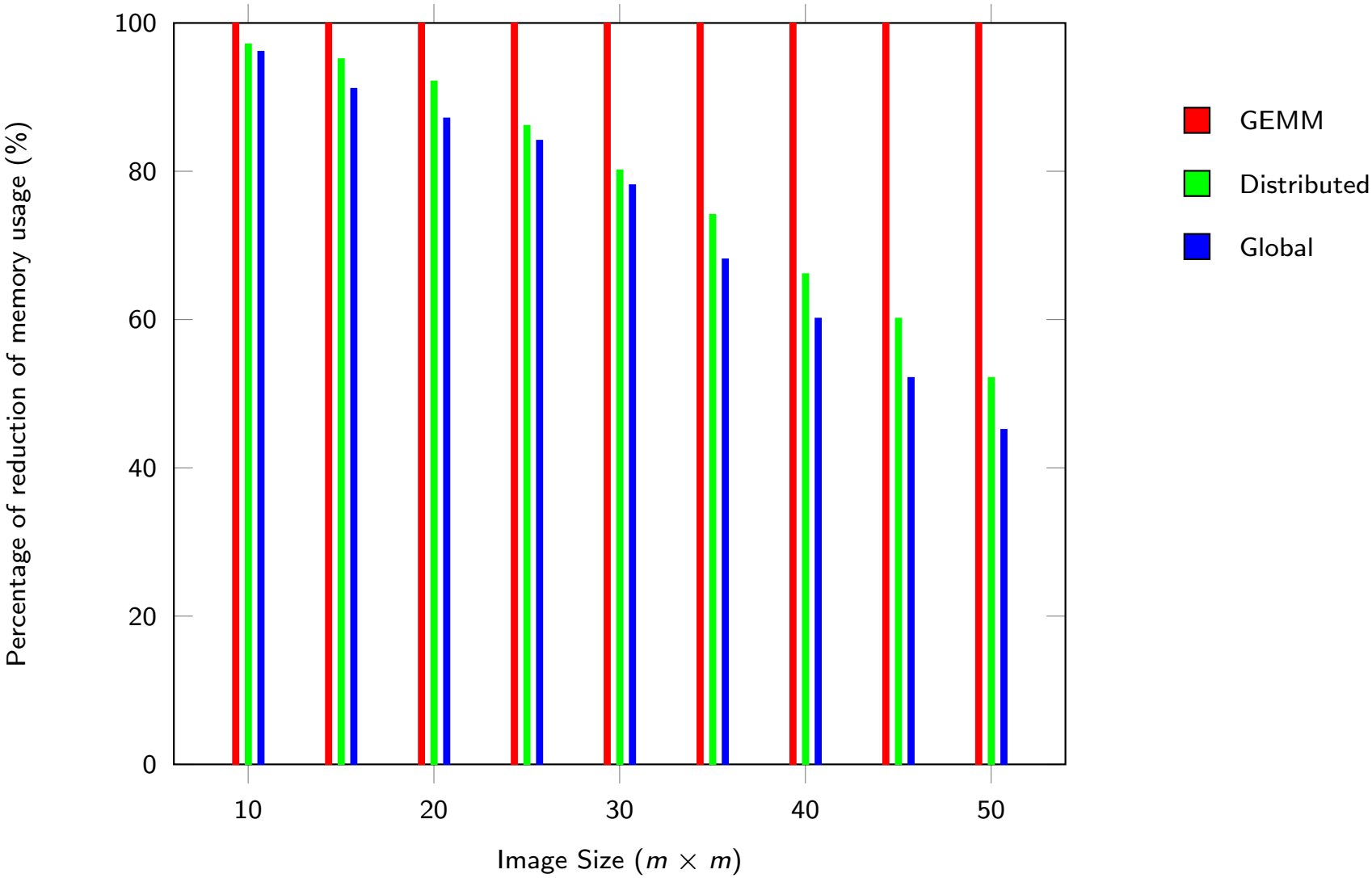
Optimize all layers together, considering dependencies and interactions. The overall execution time constraint is:

$$\sum_{l \in [1..L-1]} \mathbf{t}_p^l \leq D$$



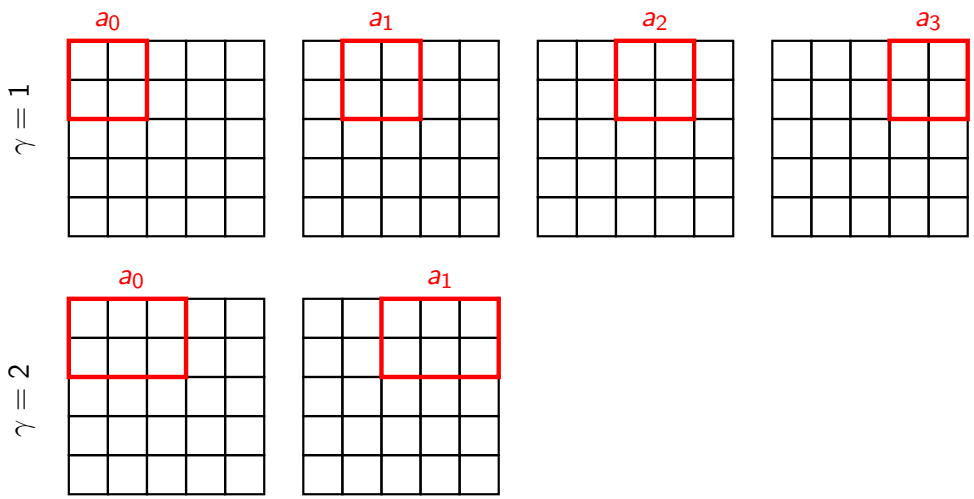
$\sum_{l \in [1..L-1]} \mathbf{t}_p^l \leq D$

Experiments

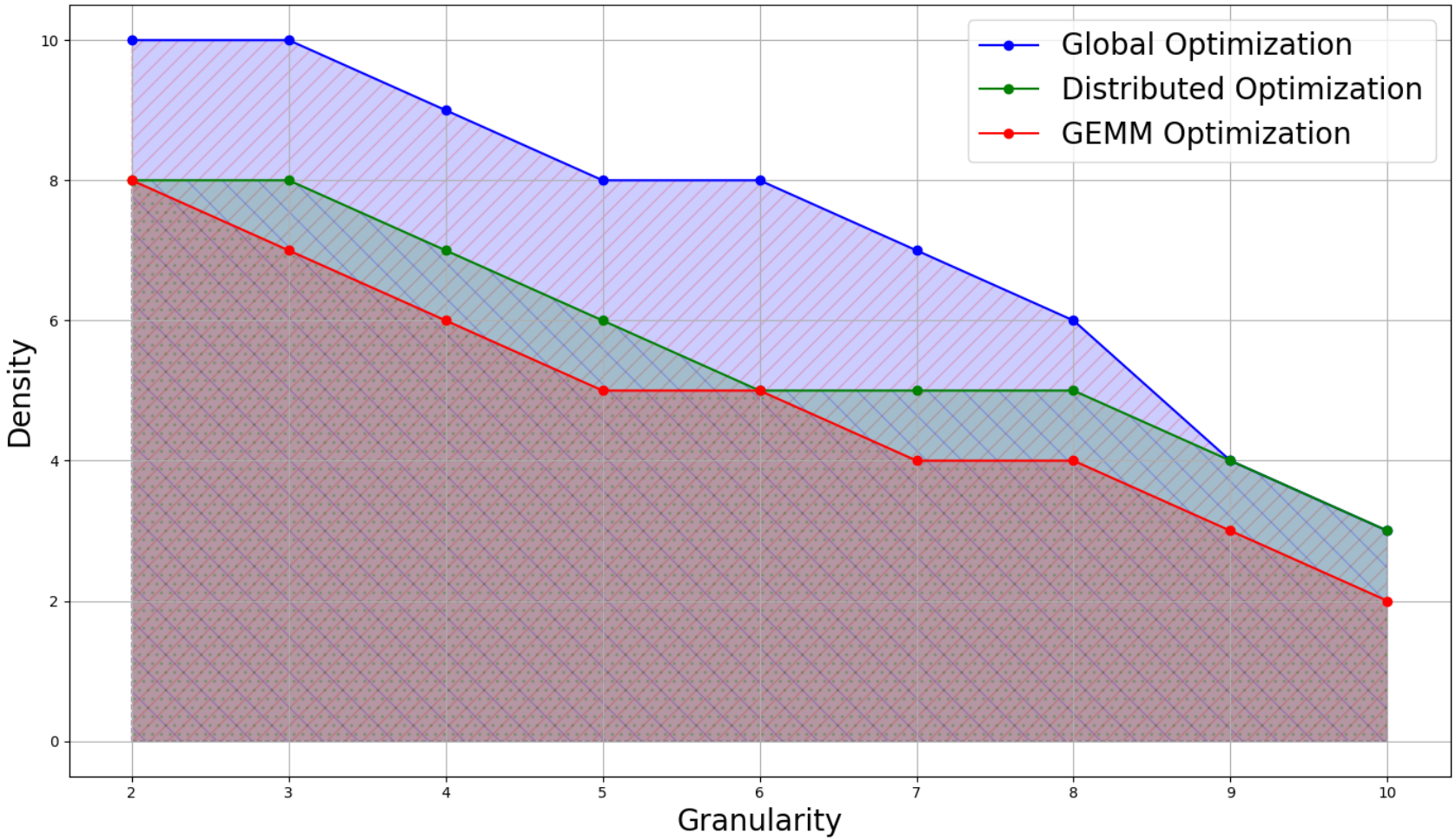


Experiments

- Granularity γ groups multiple operations into indivisible units.
- Grouped operations tend to follow a sequential pattern, as the optimizer favors adjacent pixels.
- This pattern arises from the row-major order storage of the image, where adjacent pixels are naturally grouped.



Experiments



Conclusion

- We proposed an abstract model considering the memory structure of multicore scratchpad-based architectures to minimize memory interference.
- Two implementation strategies were developed:
  - The first optimizes layers independently using intermediate deadlines.
  - The second serializes the optimization of all layers.

Future Work

- Consider handling multiple simultaneous tasks and include schedulability analysis in the optimization process
- Explore automatic task code generation
- Aim to improve communication cost estimations, potentially enhancing overall system performance
- Extend the model to other architectures by modifying the memory model, enabling its application to diverse hardware platforms

Thank you!