ARCHITECTURE-AWARE MAPPING AND SCHEDULING OF IMA PARTITIONS ON MULTI-CORE PLATFORMS

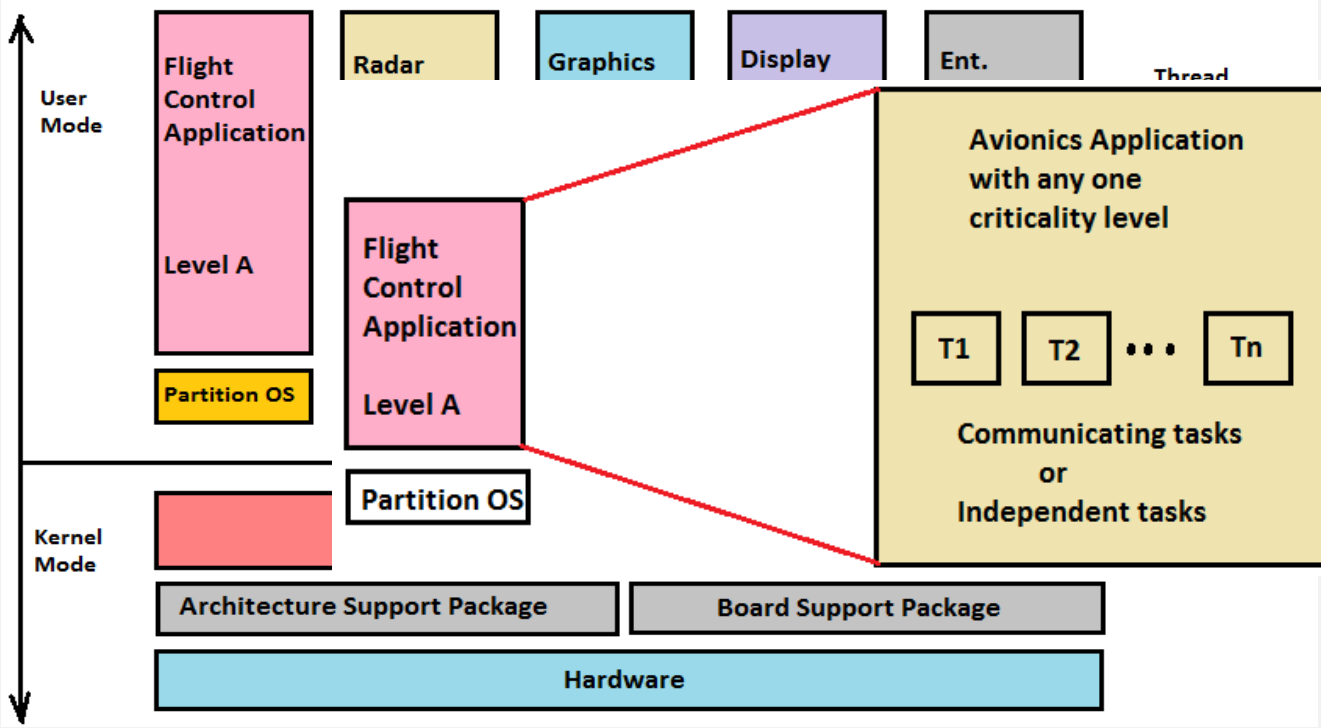
Aishwarya Vasu [1], Harini Ramaprasad [2]

[1] Southern Illinois University Carbondale

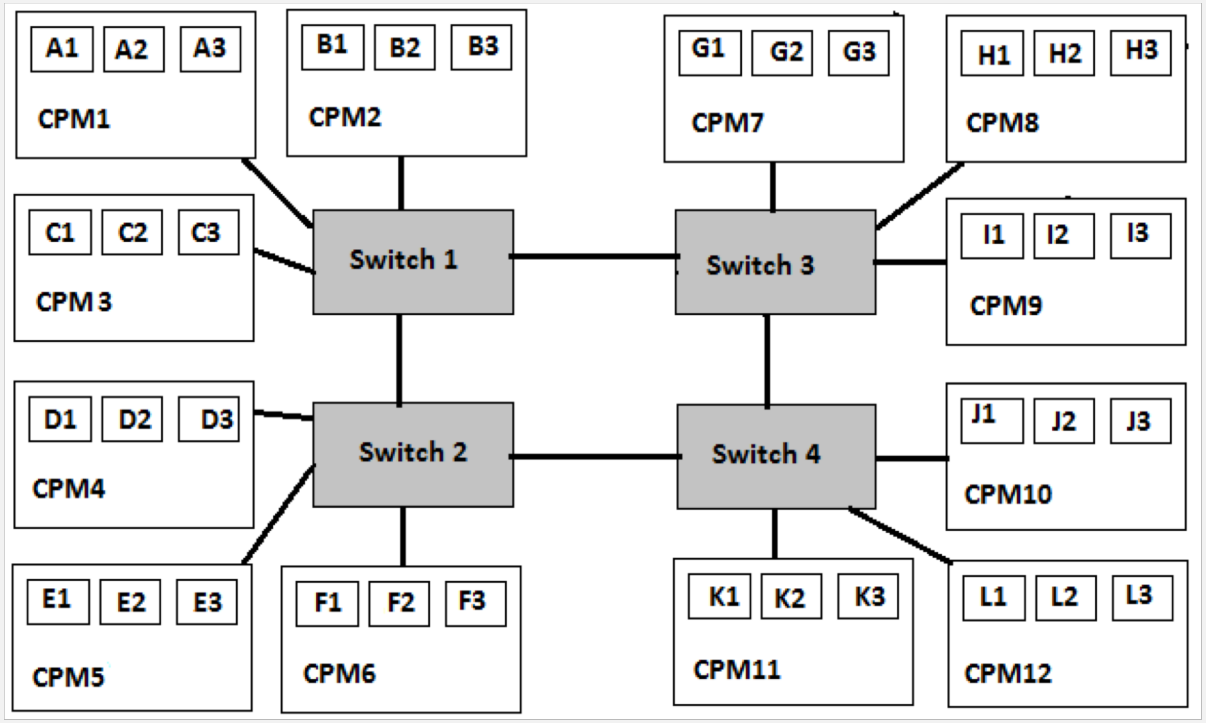
[2] University of North Carolina at Charlotte

INTEGRATED MODULAR AVIONICS

- Deploy multiple software functions with different criticality levels on single CPU



IMA PARTITIONS ON SINGLE CPU HARDWARE

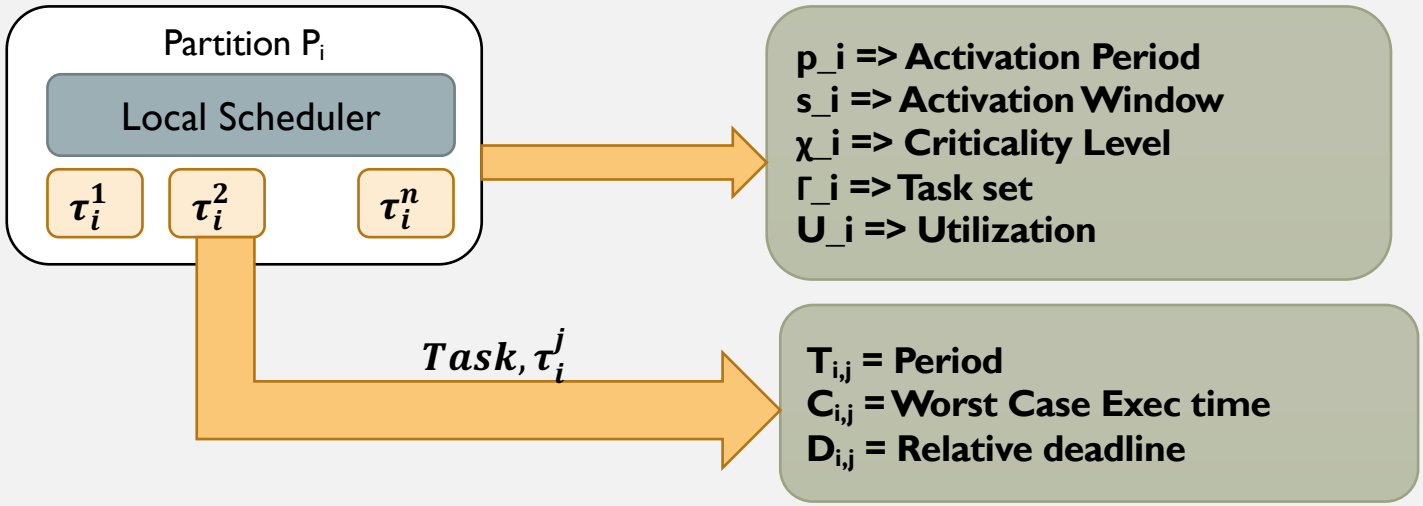


- Results in bulky system with high power consumption
- To improve Size, Weight & Power considerations
  - Deploy multiple IMA partitions on one multi-core platform

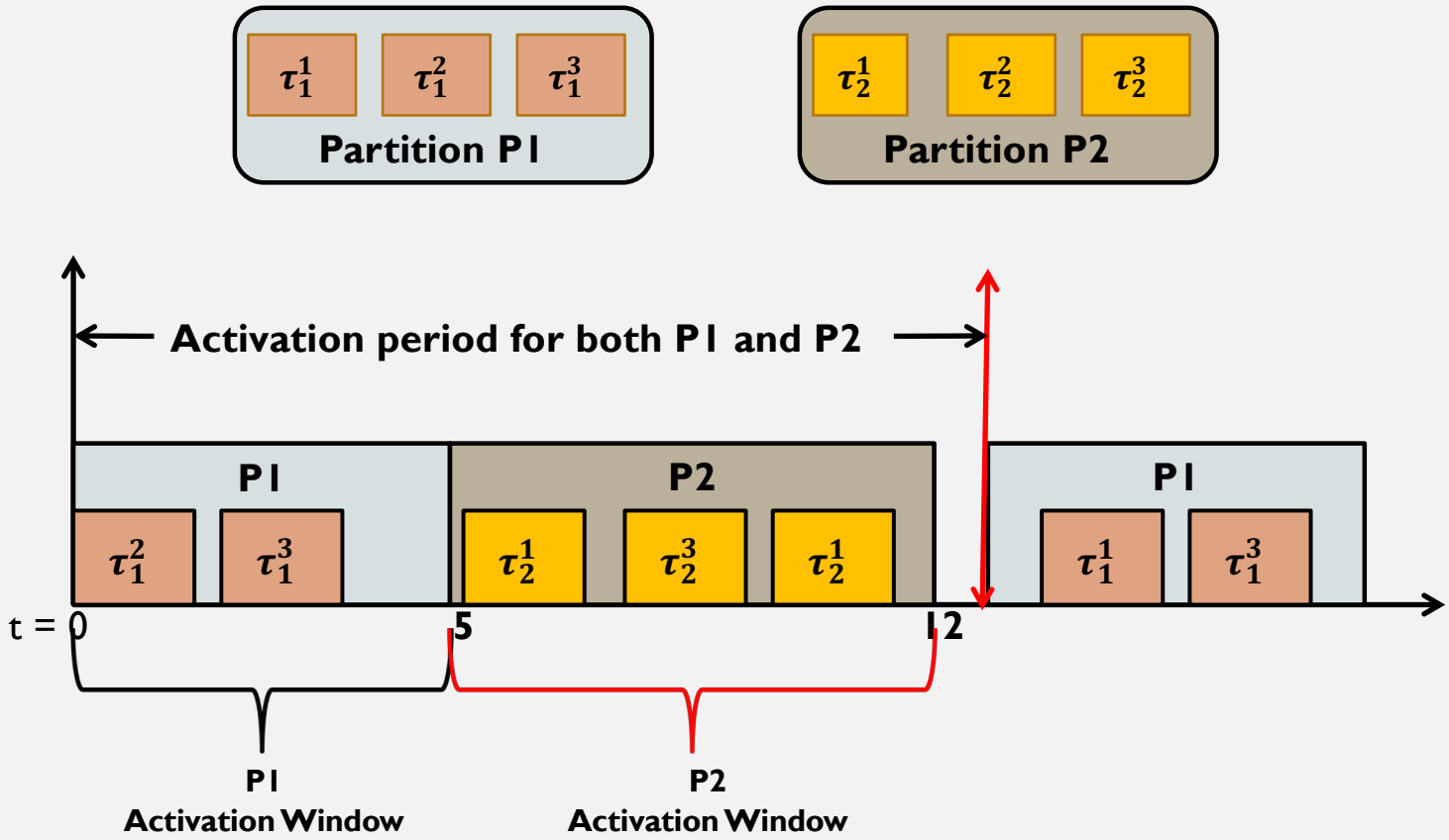
ARCHITECTURAL ASSUMPTIONS

- Identical cores
- Private data cache with support for line level locking
- Cores connect to main memory via shared bus
  - Time Division Multiple Access arbitration policy on shared bus
- Data concentrator device on each core to support asynchronous communication

PARTITION AND TASK MODEL

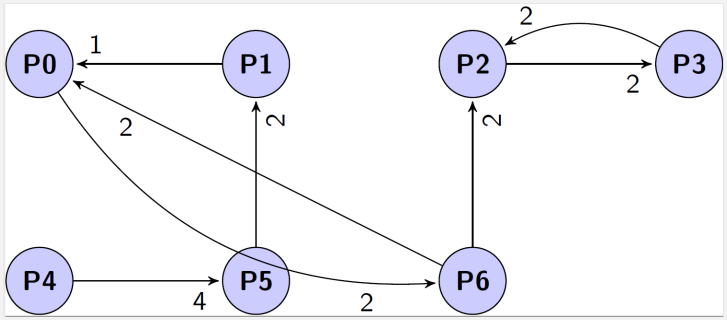


PARTITION AND TASK SCHEDULING



OBJECTIVE

- Develop algorithm to map IMA partitions onto multi-core platform when:
  - High criticality partitions may communicate (asynchronous)



Cache requirements:
{ <SA, ne, freq > }

  - High criticality partitions may load and lock specific content in core's private cache
  - Certain partition pairs cannot be allocated to the same core
    - *Partition exclusion property*
    - May Arise out of Security, Safety and Criticality Considerations or based on Risk Analysis

Provided by system integrators

ALLOCATION ALGORITHM

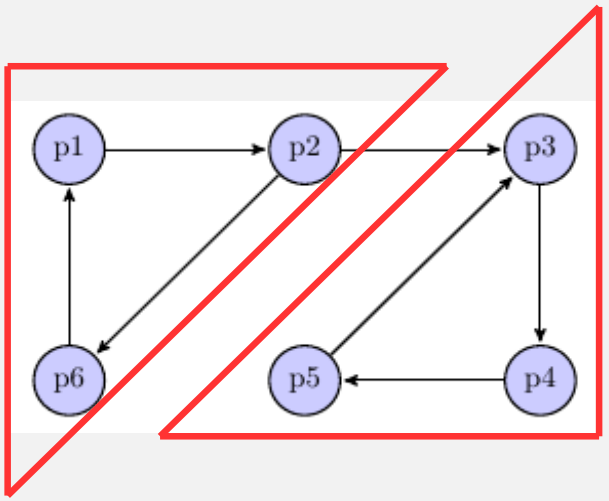
- Weight-based approach:
  - PE_i - Set of pairwise Partition Exclusion weights
    - Reflect safe or unsafe allocation of partition combinations
    - *Assumed to be provided by system integrators*
  - CO_i - Set of pairwise weights for partition P_i
    - Reflect degree of communication with other partitions
  - CA_i - Set of pairwise weights for partition P_i
    - Indicate degree of cache conflicts with other partitions
  - Resultant Weight ($\rho_{i,j}$) calculated for every partition pair P_i , P_j
    - Indicates how suitable it is to allocate P_i and P_j on same core

ALLOCATION ALGORITHM

- Two Phases:
  - Preprocessing Phase:
    - Extract & sort Strongly Connected Components (SCCs)
    - Derive pair-wise weights, core threshold weight
  - Allocation and Scheduling Phase:
    - Allocate partitions based on resultant weight between partition pairs

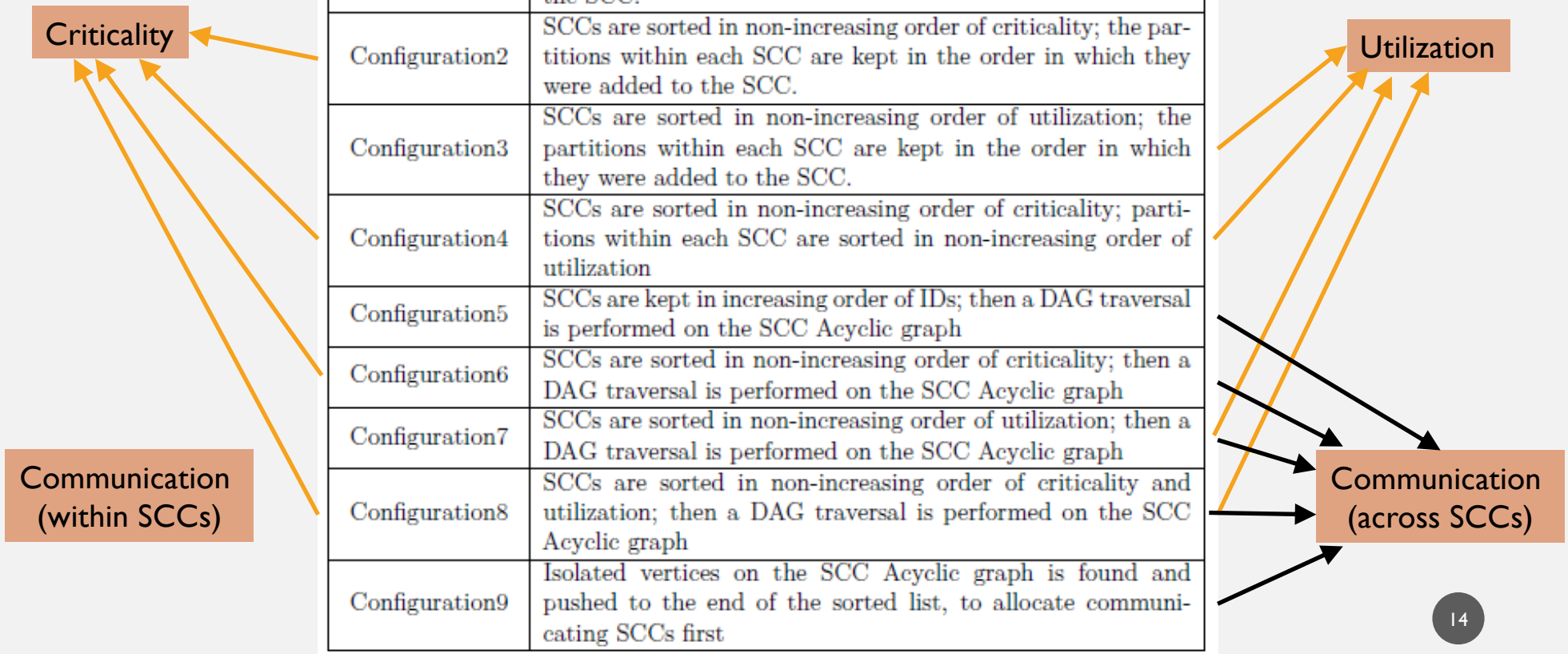
PREPROCESSING PHASE – SCC EXTRACTION AND SORTING

- Extract Strongly Connected Components (SCCs)
  - $< SCC_{id}, U_{scc}^{id}, L_{scc}^{id} >$
- Sort SCCs
  - To help in keeping communicating partitions together
  - Improves Schedulability



PREPROCESSING PHASE – SCC SORTING STRATEGY

SCC Sorting Configuration	Description
Configuration1	SCCs are kept in increasing order of IDs; the partitions within each SCC are kept in the order in which they were added to the SCC.
Configuration2	SCCs are sorted in non-increasing order of criticality; the partitions within each SCC are kept in the order in which they were added to the SCC.
Configuration3	SCCs are sorted in non-increasing order of utilization; the partitions within each SCC are kept in the order in which they were added to the SCC.
Configuration4	SCCs are sorted in non-increasing order of criticality; partitions within each SCC are sorted in non-increasing order of utilization
Configuration5	SCCs are kept in increasing order of IDs; then a DAG traversal is performed on the SCC Acyclic graph
Configuration6	SCCs are sorted in non-increasing order of criticality; then a DAG traversal is performed on the SCC Acyclic graph
Configuration7	SCCs are sorted in non-increasing order of utilization; then a DAG traversal is performed on the SCC Acyclic graph
Configuration8	SCCs are sorted in non-increasing order of criticality and utilization; then a DAG traversal is performed on the SCC Acyclic graph
Configuration9	Isolated vertices on the SCC Acyclic graph is found and pushed to the end of the sorted list, to allocate communicating SCCs first



PREPROCESSING PHASE – DERIVATION OF CO_I

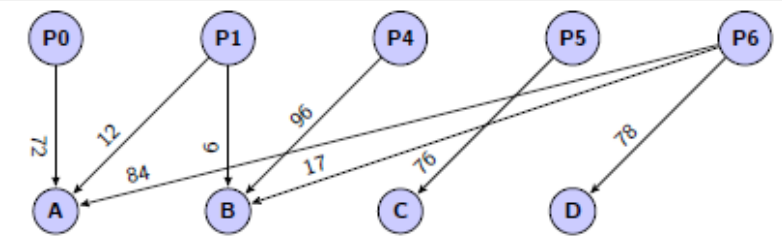
- Define Communication Weight between partition pairs:
  - $CO_{ij} = < co_{ij}, cost_{ij} >$
  - $co_{ij} = \begin{cases} 1, & if\ Pi, Pj\ communicate \\ 0, & otherwise \end{cases}$

$$cost_{i,j} = \left\lceil \frac{n_{i,j}}{n_{i,j}^{trans}} \right\rceil * m_{tx}^{latency}$$

  - $n_{i,j}$: number of bytes transferred from partition P_i to P_j
  - $n_{i,j}^{trans}$: number of bytes transferred per transaction
  - $m_{tx}^{latency}$: communication latency incurred per transaction

PREPROCESSING PHASE – DERIVATION OF CA_I

- Bipartite graph constructed
  - Partitions on top
  - Groupings of cache sets on bottom
  - Edge weight
    - Represents number of cache lines that partition tries to lock in that group of cache sets



- A partition pair cannot have cache conflict if one of two conditions is satisfied:
  - No cache set that both partitions try to lock
  - Every cache set that both partitions try to lock has less incoming edges than capacity of set
- Cache Conflict Weight

$$CA_{i,j} = \frac{(Lines_{total} - Lines_{i,j}^{Conflict})}{Lines_{total}}$$

  - $Lines_{total}$: Total number of lines in cache
  - $Lines_{i,j}^{conflict}$: Number of conflicting lines in cache for P_i and P_j

ALLOCATION PHASE - OVERVIEW

- Goal: Find number of cores needed to allocate partition set

- Two Schemes
  - NCU Scheme:
    - Strict consideration of Communication, PE and Cache requirements
    - Partitions with potential cache conflicts allocated on different cores
  - CU Scheme:
    - Consideration of Communication and PE requirements
    - Cache requirements relaxed → allow conflicting partitions on same core if needed
      - Subset of conflicting lines are *unlocked* by one partition
      - Results in increase of utilization

ALLOCATION PHASE – HIGH CRITICALITY PARTITION ALLOCATION

- Allocate High Criticality Partitions based on weights
  - Define Core Threshold Weight, Ω
    - Based on recommended weight for individual factors (provided by system integrators)
  - Partition pairs with resultant weight $\rho_{i,j}$ >= Ω can be allocated on same core
  - For every partition:
    - Compute resultant weight on all cores (i.e., try allocating partition on each core)
    - Get information on *actual* cache conflicts
    - Remove cores with resultant weights less than Core Threshold Weight, Ω
    - Sort remaining cores in non-increasing order of resultant weights

ALLOCATION PHASE – HIGH CRITICALITY PARTITION ALLOCATION

- Iterate over sorted cores
  - Compute communication costs if needed
  - Check schedulability of partitions that had change in utilization due to communication
    - Compute activation window, activation period
    - Based on an existing work in hierarchical scheduling
  - If successful, allocate partition to core and end iteration
- If core not found, next steps depend on CU / NCU scheme

Alejandro Masrur, Thomas Pfeuffer, Martin Geier, Sebastian Drössler, and Samarjit Chakraborty. 2011. "Designing VM schedulers for embedded real-time applications", In Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. ACM, 29–38.

ALLOCATION PHASE – HIGH CRITICALITY PARTITION ALLOCATION

- NCU Scheme:
  - "Add" new core to system
  - Allocate partition to new core if possible after accounting for communication costs
- CU Scheme:
  - Compute cache conflict latency for all partitions conflicting with P_i
    - Update Partition utilization
  - Sort cores in non-decreasing order of their change in utilization
  - Re-try cores and check schedulability
- If no core found
  - P_i deemed to be non-schedulable
  - Cache unlocking and utilization changes are reverted to previous values

ALLOCATION PHASE – LOW CRITICALITY PARTITIONS

- Allocated using Worst-Fit heuristic
- Sort partitions in non-increasing order of criticality and utilization
- For every partition P_i
  - Sort cores in non-increasing order of available utilization
  - Try core with maximum available utilization
  - "Add" new core if core with maximum available utilization cannot fit partition P_i

SIMULATION SETUP – PARTITIONS & TASKS

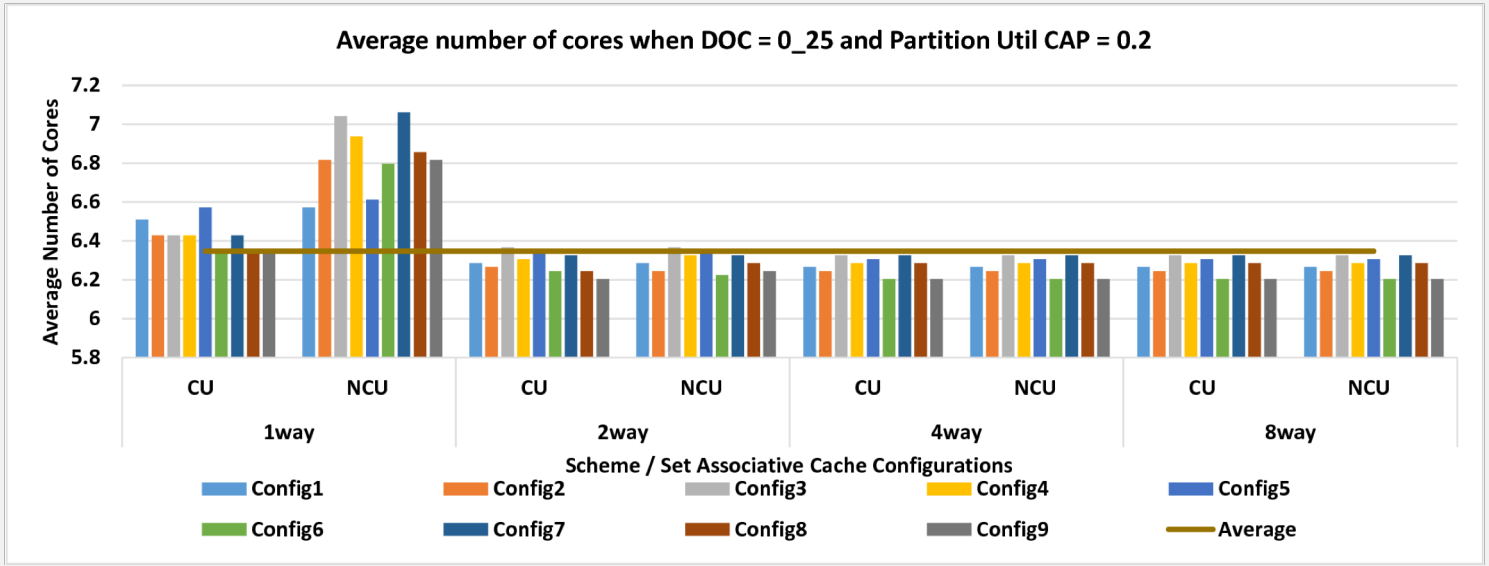
- Multiple partition utilization caps - 0.2, 0.3, 0.4, 0.5, 0.6 - considered
- For each cap, 100 sets of different partition and task characteristics generated
- Random directed weighted cyclic graph generated for communication between high criticality partitions
  - Degree of Communication (DoC): (0% - 25%), (25% - 50%)
- Random memory footprints generated for high criticality partitions
- Random Partition Exclusion weights generated between high criticality partitions

SIMULATION SETUP – ARCHITECTURAL DETAILS

- Identical cores
- Private data cache on each core

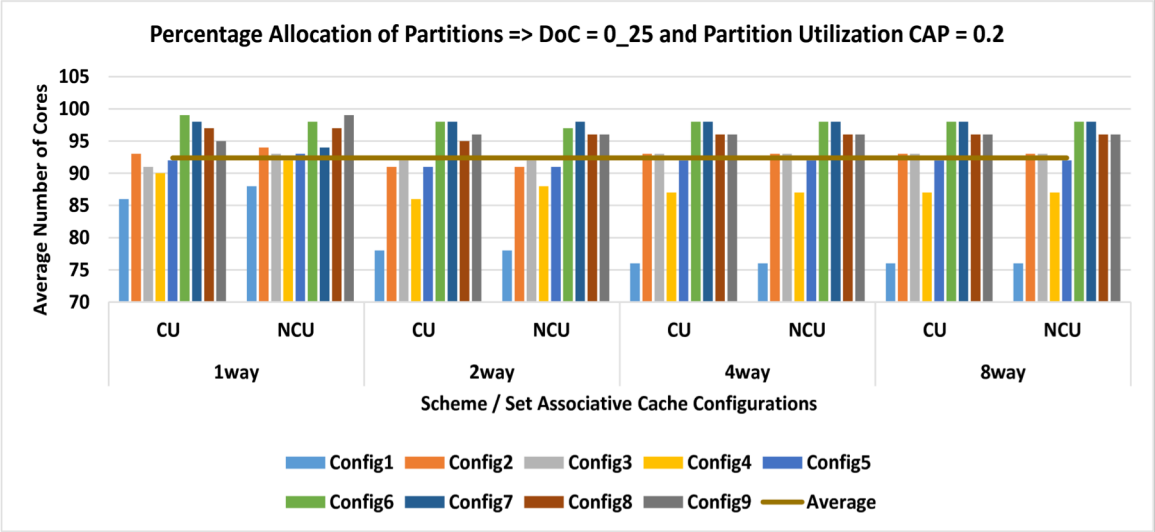
Parameter	Size
Cache line size	32 B
Element size	16 B
Associativity	1 (32 KB)
	2 (64 KB)
	4 (128 KB)
	8 (512 KB)
	16 (1 MB)
Memory Access latency	50 cycles

COMPARISON OF AVERAGE NUMBER OF CORES BETWEEN NCU AND CU SCHEMES: DOC = (0%-25%): UTIL CAP = 0.2



- NCU
  - More cores required to host partitions for 1 way set-associative cache configuration
  - Reason: increased number of cache conflicts
- CU Scheme tries to accommodate partitions by unlocking conflicting cache lines
  - Uses a less number of cores when compared to NCU scheme
- When cache ways are increased, average number of cores decreases
  - Reason: reduced number of cache conflicts

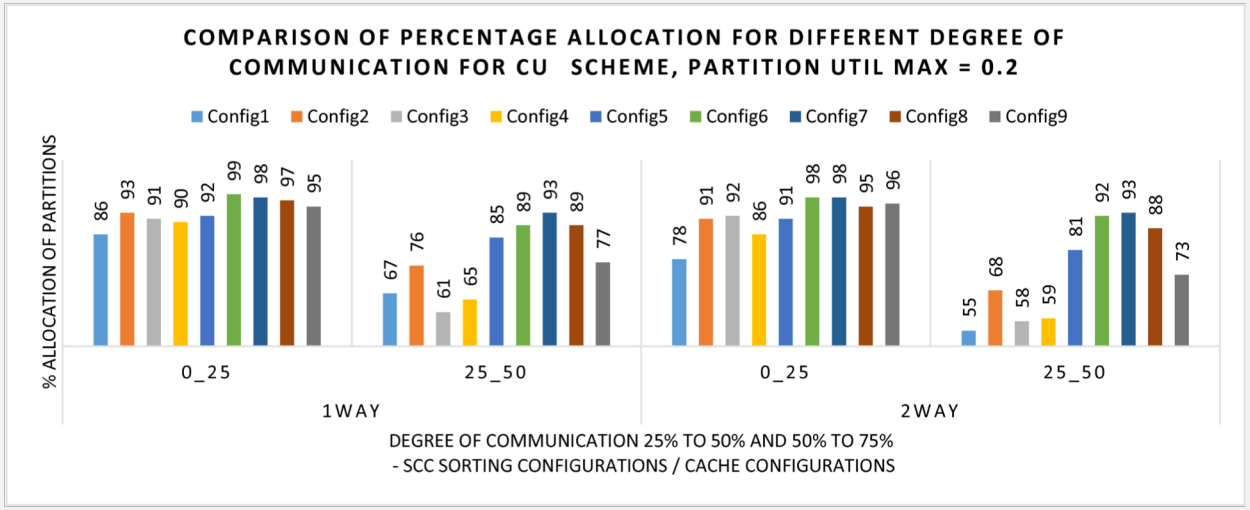
COMPARISON OF PERCENTAGE ALLOCATION OF PARTITION SETS BETWEEN CU AND NCU SCHEMES



- For lower $\widehat{U}$, (0.2, 0.3 and 0.4)
  - Configs 1 - 4 schedule lower percentage of partition sets than Configs 5 - 9
  - Configs 1 - 4 do not keep communicating partitions together unless they are within same SCC
- Beyond 1way cache configuration, no significant difference between performance of CU & NCU schemes
  - Although there are potential cache conflicts between partitions, not all of them manifest as actual conflicts even in NCU scheme

EFFECT OF DEGREE OF COMMUNICATION ON ALLOCATION – CU SCHEME: COMPARISON BETWEEN DOC = 0_25% AND DOC = 25_50%

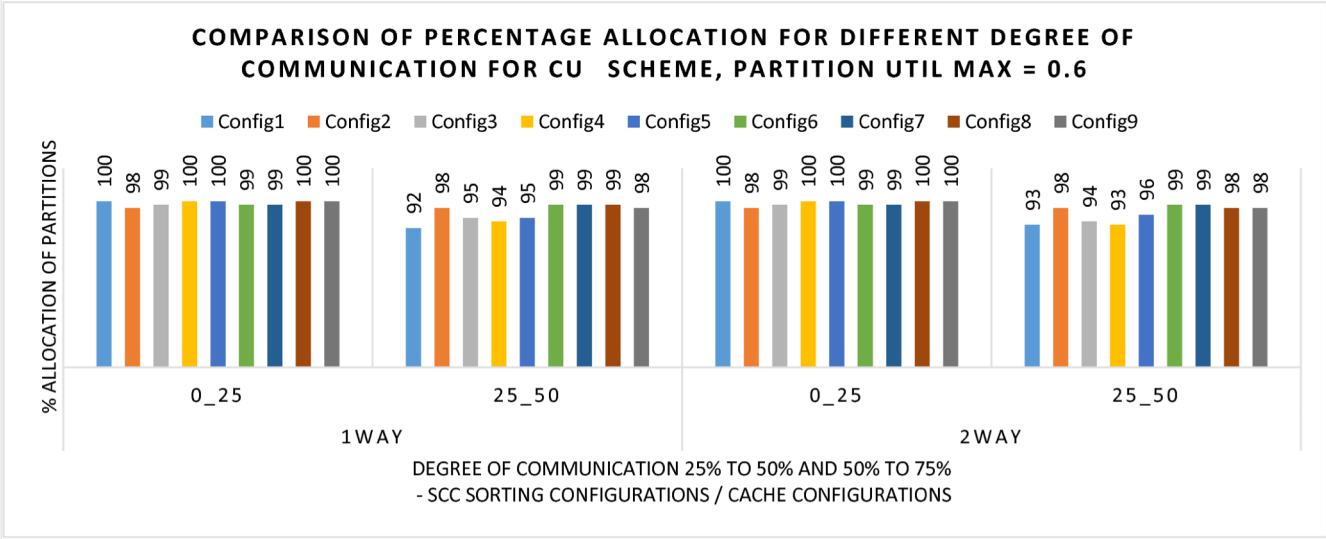
Partition Utilization cap = 0.2



- As DoC is increased, % of successfully allocated partition sets decreases
- Change in % allocation with increased communication is higher for lower $\widehat{U}$
  - More number of partitions for lower $\widehat{U}$ => more communicating partitions => increased communication cost

EFFECT OF DEGREE OF COMMUNICATION ON ALLOCATION – CU SCHEME: COMPARISON BETWEEN DOC = 0_25% AND DOC = 25_50%

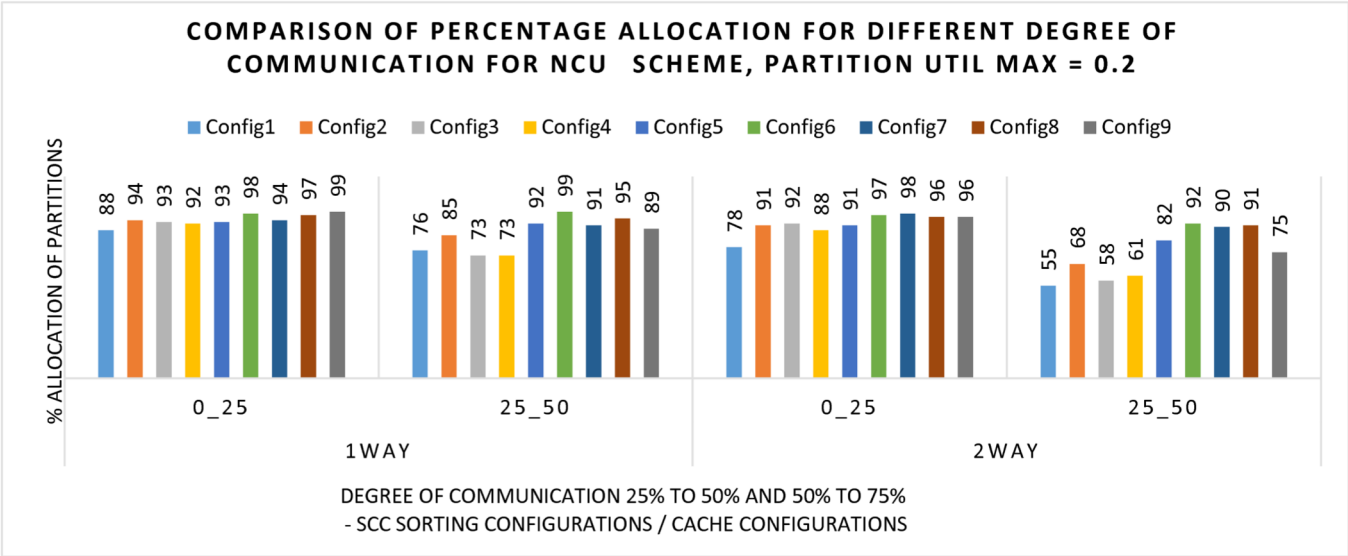
Partition Utilization cap = 0.6



- As $\widehat{U}$ increases
  - Lower number of partitions in a set => Lower communication => DoC less significant

EFFECT OF DEGREE OF COMMUNICATION ON ALLOCATION – NCU SCHEME: COMPARISON BETWEEN DOC = 0_25% AND DOC = 25_50%

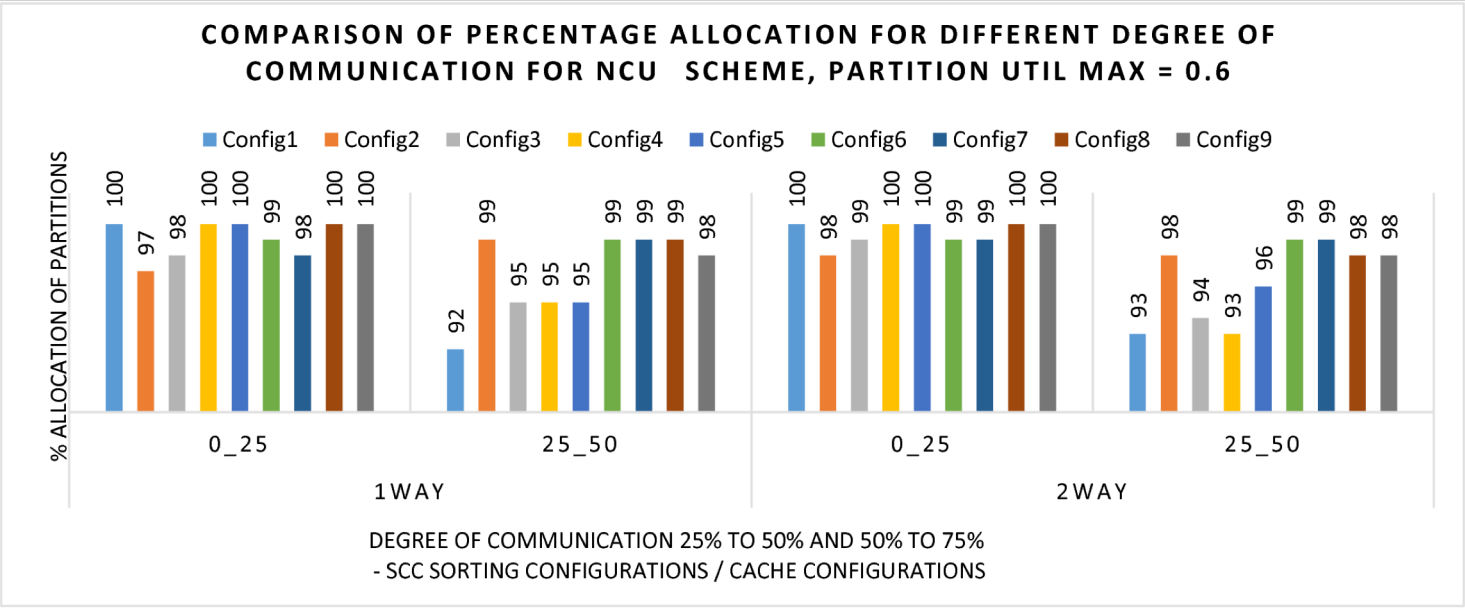
Partition Utilization cap = 0.2



- Similar trend observed for NCU scheme

EFFECT OF DEGREE OF COMMUNICATION ON ALLOCATION – NCU SCHEME: COMPARISON BETWEEN DOC = 0_25% AND DOC = 25_50%

Partition Utilization cap = 0.6



- Similar trend observed for NCU scheme for higher $\widehat{U}$

CONCLUSIONS AND FUTURE WORK

- Outcome → design space exploration tool – useful during system integration phase
- **Allocation of partitions is impacted by:**
  - Order in which partitions are chosen for allocation
  - Degree of Communication (DoC) among partitions
- **Future Work:**
  - Enhance cache conflict generator to conduct sensitivity studies and observe how increasing conflicts affect our algorithm's performance
  - Consider allocation and scheduling of partitions that share software resources