Duplicated execution method for NoC-based Multiple Processor Systems with Restricted Private Memories

Masashi Imai

(miyabi@hal.rcast.u-tokyo.ac.jp)

Background

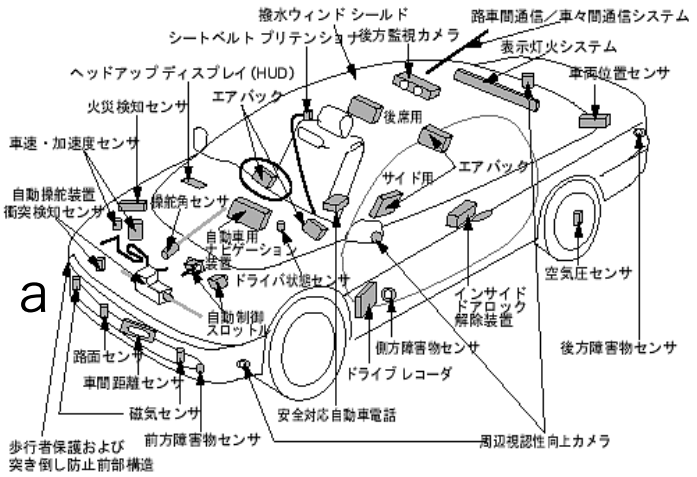
- VLSI technology scaling
  → The performance improvement of a single processor is limited due to clock skew, power dissipation, ILP, and complexity
- CMP (Chip Multi-Processor) and MPSoC (Multiple-Processor System-on-a-Chip)
  - Promising architecture, not only for high performance but also for dependability
    - Even if a processor core becomes faulty, the remaining cores can continue to operate
- A simple bus architecture does not scale with the system size as the bandwidth is shared by all the cores attached it
- On-chip network is a feasible solution to many-core systems

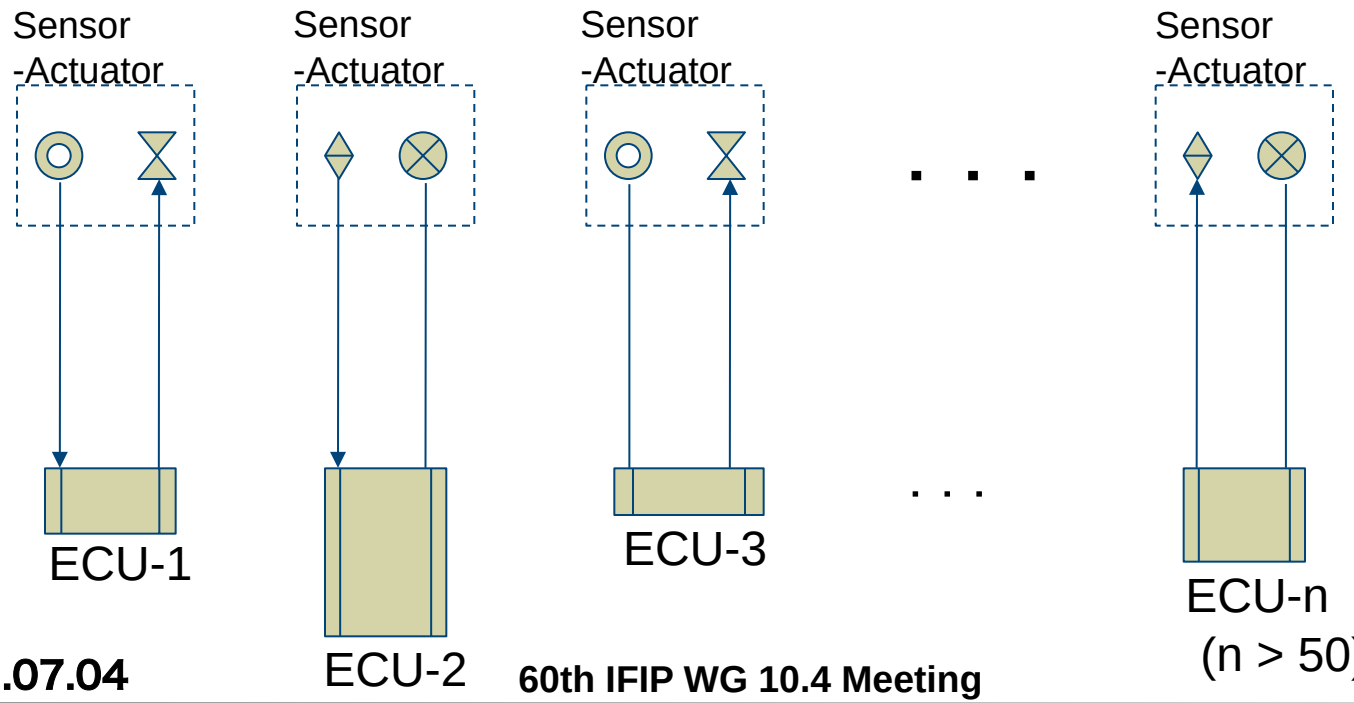
→ Focus on NoC-based multiple processor core architecture

- "Dependable Network-on-Chip Platform" project funded by CREST @JST (Japan Science and Technology Agency)

Target applications

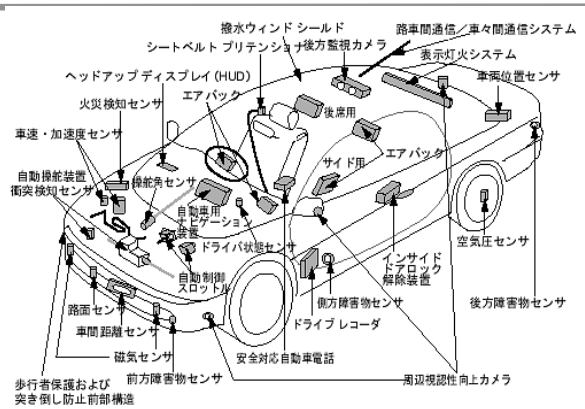
- Automotive engine control
  - Various periodic tasks control actuators in response to the corresponding sensor inputs in a deadline period (ex. 4msec)
- Conventional methods

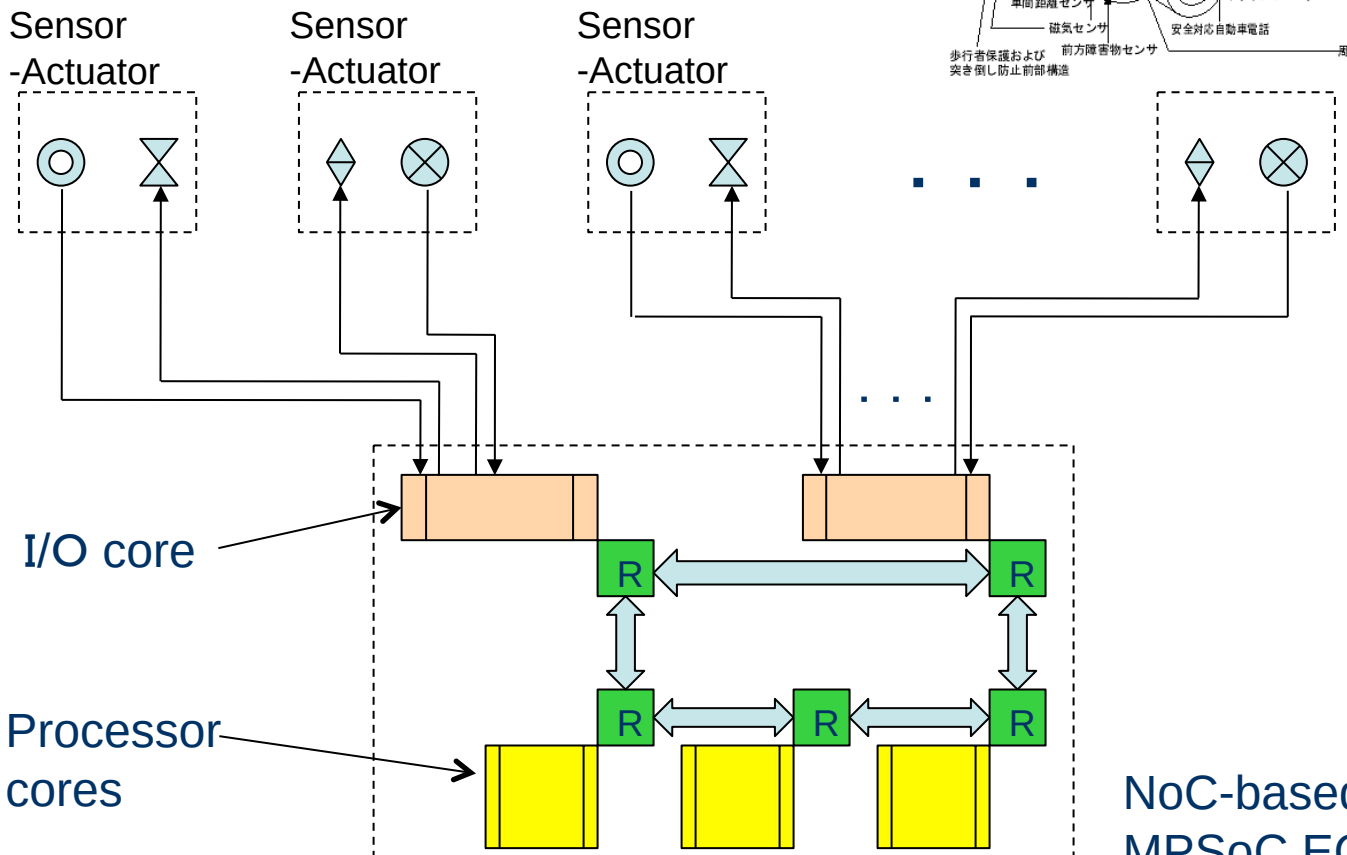




Target applications

- Automotive engine control
- NoC-based MPSoC approach





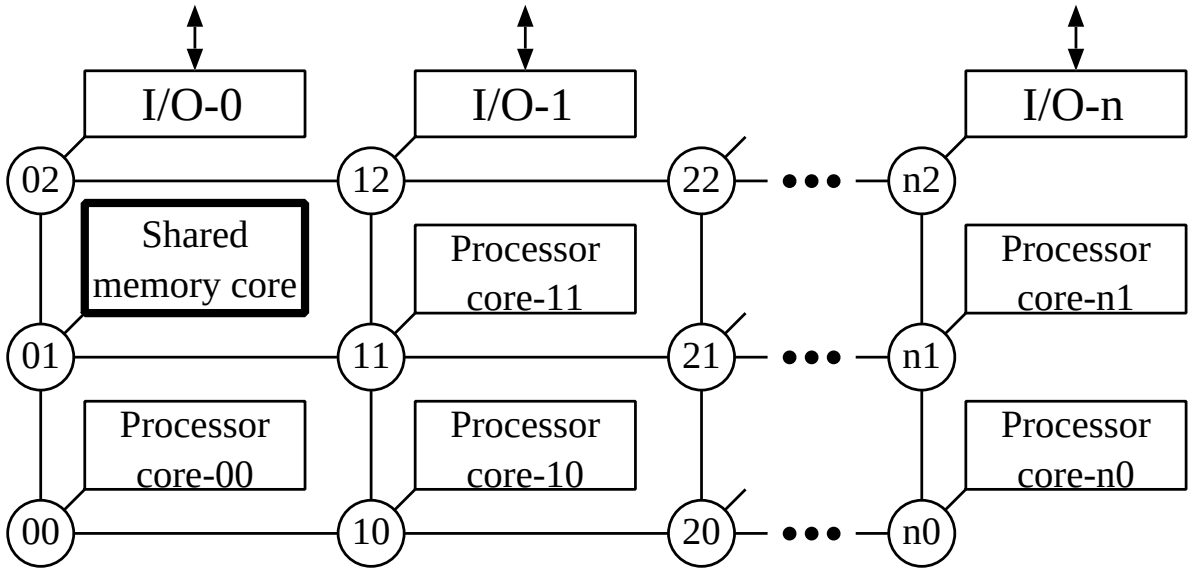
NoC-based
MPSoC ECU

Task allocation and scheduling

- Task allocation and scheduling method is a main issue for performance optimization
  - Dynamic task allocation methods
    [E.Carvalho,RSP07][C.L.Chou,DATE08][T.Wei,ISQED10]
    [K.Lakshmanan,RTSS10]…
  - Static task allocation methods
    [J.Hu,Trans.CAD05][C.Marcon,ASP-DAC05] …
- These allocation methods depend on memory architecture in on-chip networks
  - Single shared memory
  - Multiple private memories

Task allocation and scheduling

- Single shared memory
  - All the processor cores can perform all the tasks
  - Requires a high speed memory interface and a mechanism to keep cache coherence
  - → Not suited for embedded systems like automotive control systems



Task allocation and scheduling

- Multiple private memories
  - Mechanism is very simple
  - Each processor core can only perform a restricted number of tasks
  - → Suited for embedded systems
    - The amount of tasks in embedded systems is small

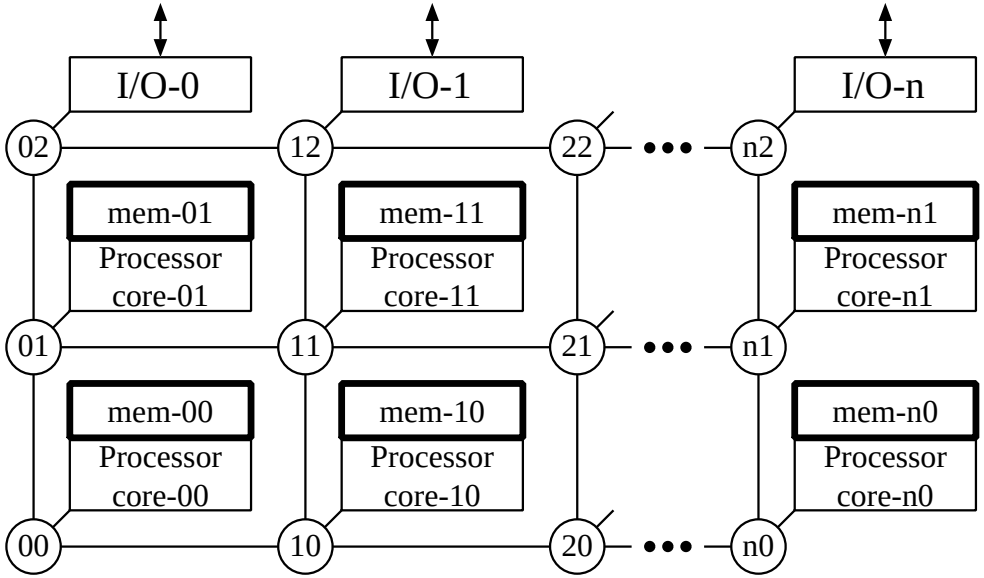
Objectives

- Target system: NoC-based MPSoC system where each processor core has its small private memory
- Propose an effective static task allocation method in order to improve the dependability of target systems
  - Each task is quadruplicated and statically assigned to private memories
  - The performance improvement is evaluated using MTTF (Mean Time To Failure)

NoC-based MPSoC

Architecture overview

- 2D mesh NoC



- I/O cores connect to routers where y=2
  - Gather sensor inputs and assign tasks to processor cores
- Processor cores connect to routers where y=0,1
  - Each processor core has its private memory

Fault model

- Single-core fault model
  - A fault is only capable of occurring in a single processor core of an NoC-based MPSoC at any one moment in time
- Permanent fault
  - We must identify the faulty core and stop using it
- Transient fault
  - The core in which a transient fault occurs can be recovered by re-executing from the latest checkpoint

  → We do not have to stop using it immediately

- There is no fault in I/O cores

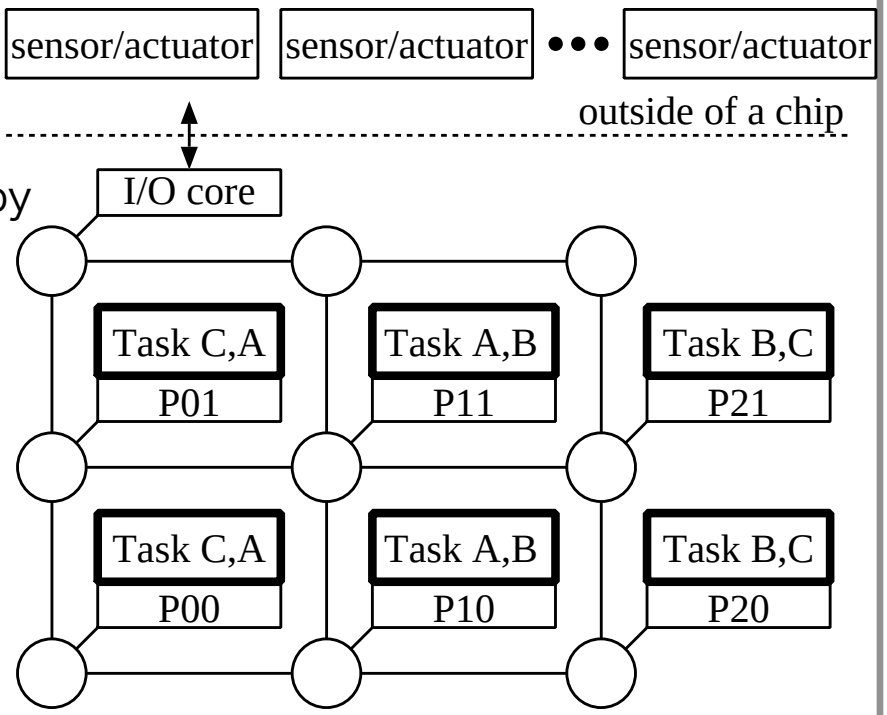
  (For simple evaluation in this presentation)

Fault detection and recovery

- Several studies have been made using CMPs to detect faults and recovery
  - mSWAT: software-based HW and SW fault detection and diagnosis method
  - DCC (Dynamic Core Coupling): processor-level fault tolerance technique
    - Employ a TMR using hot spares in order to isolate a failure core and recovery its task
  - Pair & Swap: processor-level fault tolerance technique
    - Pair phase: replication and comparison
    - Swap phase: swap and retry (retry and decision)
  - etc, ……
- In this research, fault detection and recovery are performed based on the pair & swap scheme

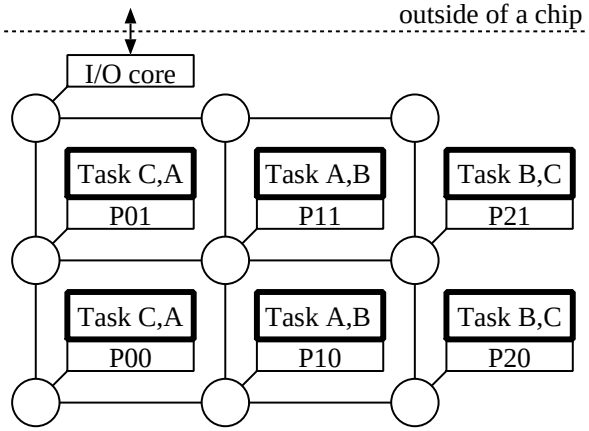
Basic mechanism

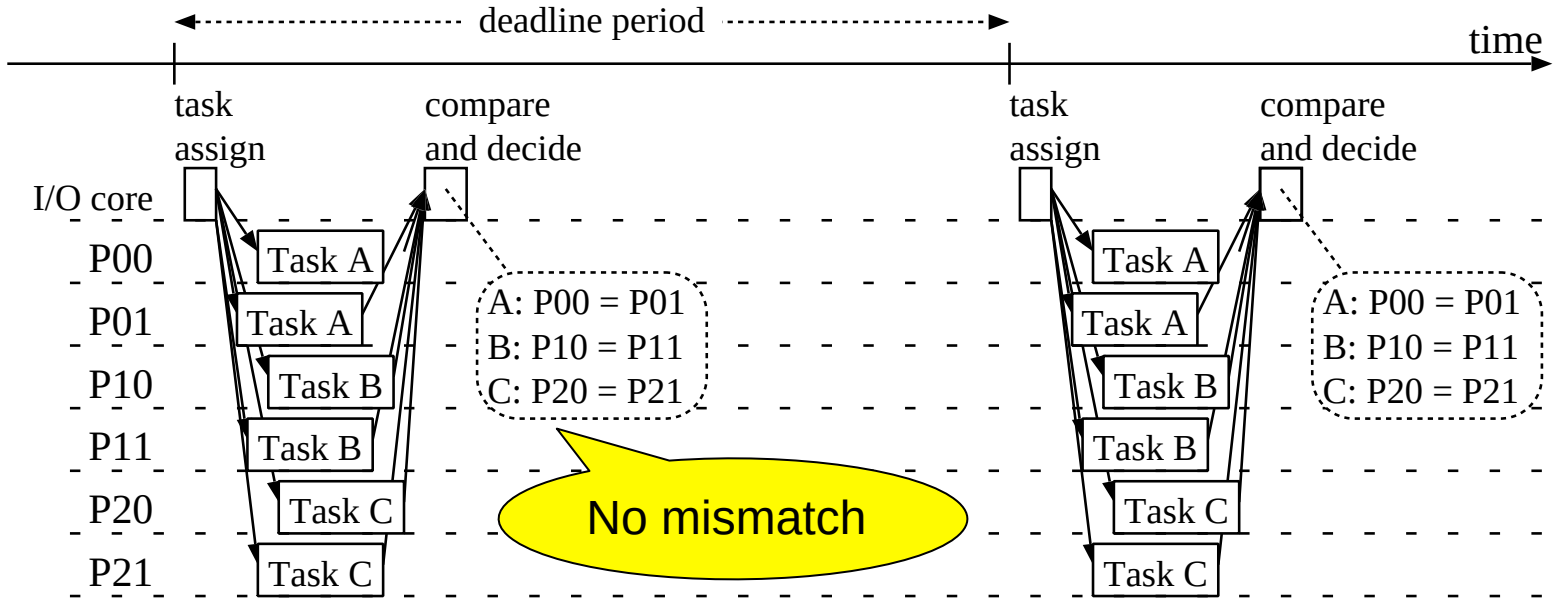
- Proposal: Each task is quadruplicated and statically assigned to private memories so that each private memory has only two different tasks
  - Ex. 6 core system
  - Task A, B, and C
    - Task A can be performed by P00,P01, P10,P11
    - Task B: P10,P11, P20,P21
    - Task C: P20,P21, P00,P01
  - Compose three pairs in order to detect faults by comparison of computation results



Basic mechanism

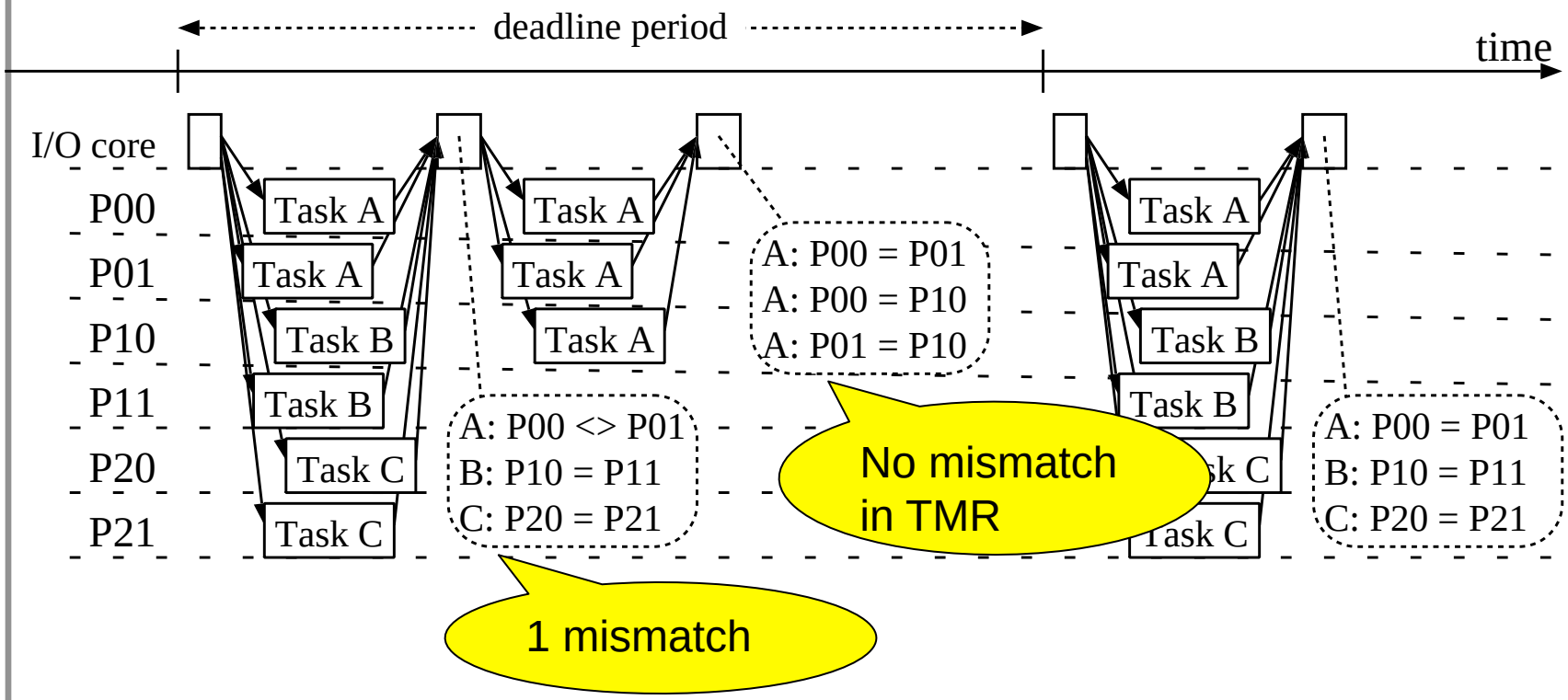
- Non faulty operation





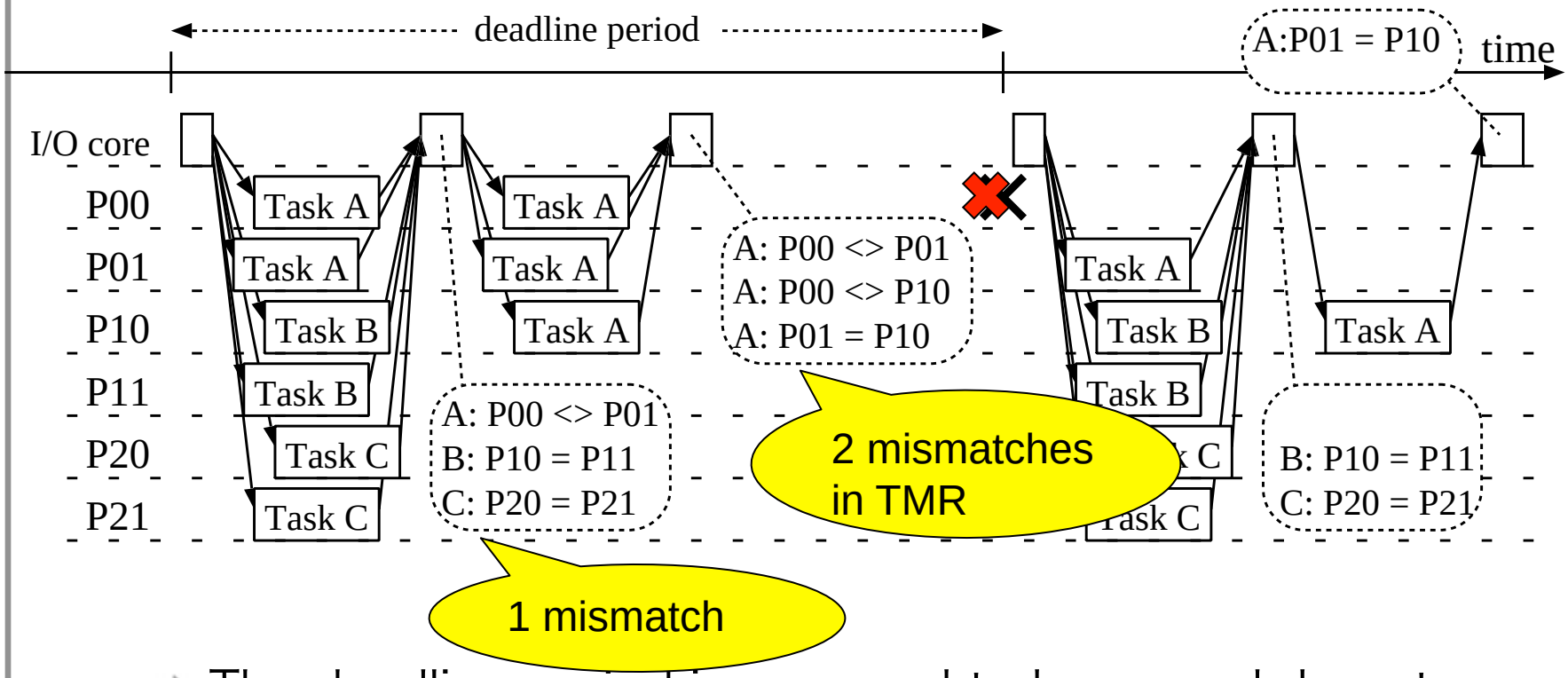
Fault location mechanism

Transient fault operation



Fault location mechanism

Permanent fault operation



- The deadline period is assumed to be enough long to execute three tasks sequentially

→ Performance degradation does not occur

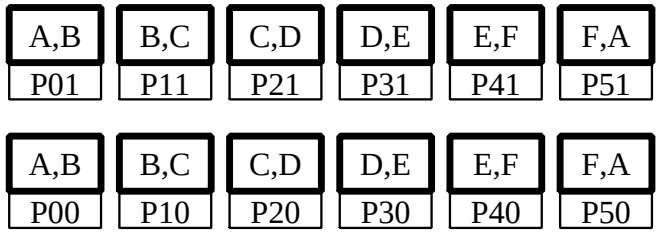
Fault location mechanism

- Features
  - Require at most three sequential task executions in the deadline period in order to diagnose the type of faults and identify a faulty processor core
  - Transient faults can be completely masked without performance overhead
  - The entire system can achieve graceful degradation
    - In the successful case, it can be allowed that N processor cores get failed in a 2N core NoC-based MPSoC

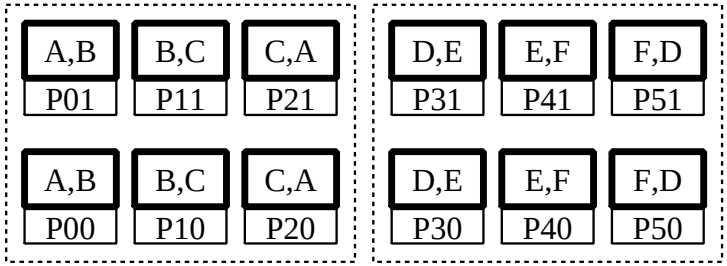
Static task allocation method

- The number of tasks is larger than 3, various configurations can be considered
- Ex. 6 tasks; task A,B,C,D,E, and F

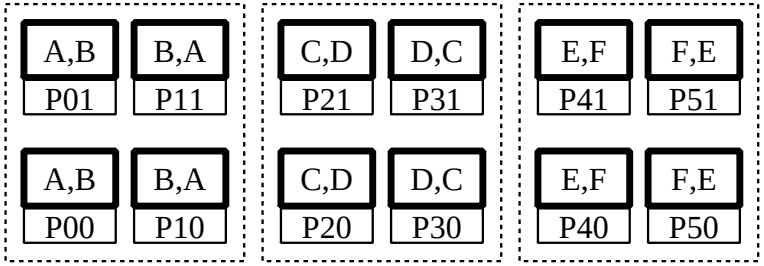
1) 6-cyclic

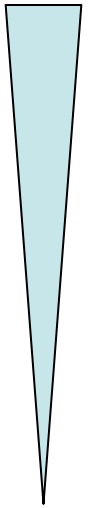


2) 3-cyclic x2



3) 2-cyclic x3





Evaluation (1)

- Research question: How to statically allocate tasks to each private memories in order to improve performance?
  - Coarse-grain configuration?
  - Fine-grain configuration?
- The performance improvement is evaluated using MTTF (Mean Time To Failure)

Comparison result (1)

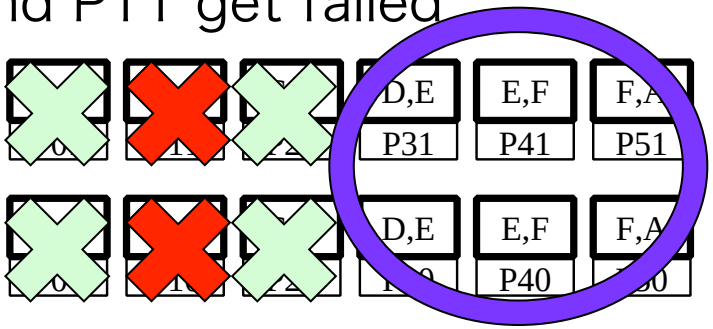
- No memory faults, 6 tasks
- The failure rate of a processor core: λ

Configuration	System MTTF
6-cyclic	0.511 / λ
3-cyclic x 2	0.507 / λ
2-cyclic x 3	0.596 / λ

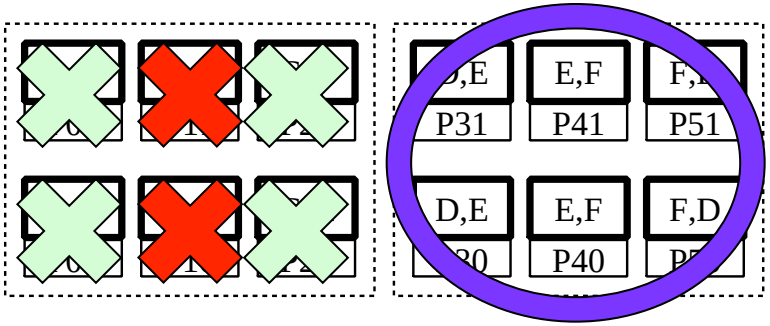
→ Fine-grain (2-cyclic) configuration is effective!

Static task allocation method

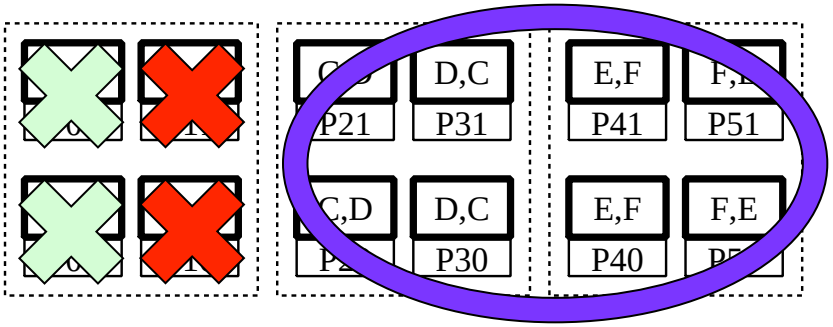
- When P10 and P11 get failed
- Cyclic-6



- Cyclic-3 x2

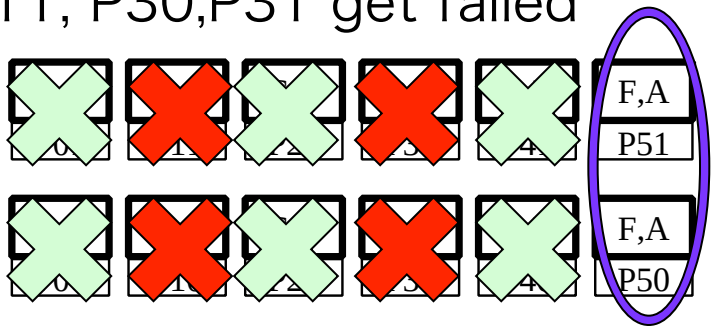


- Cyclic-2 x3

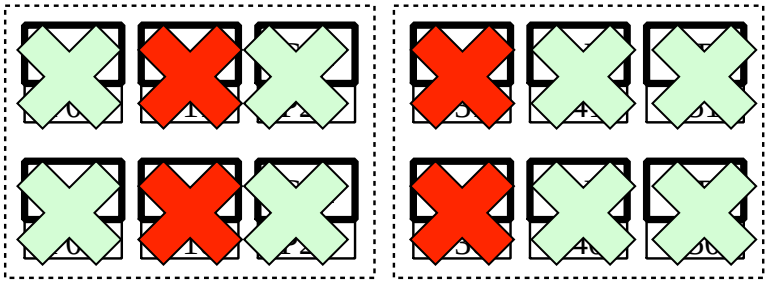


Static task allocation method

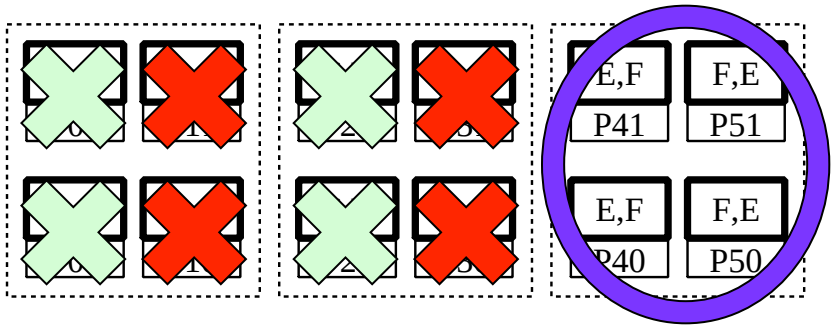
- When P10,P11, P30,P31 get failed
- Cyclic-6



- Cyclic-3 x2

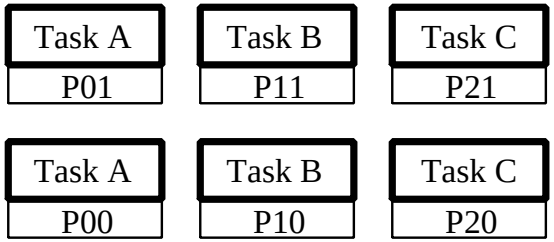


- Cyclic-2 x3



Evaluation (2)

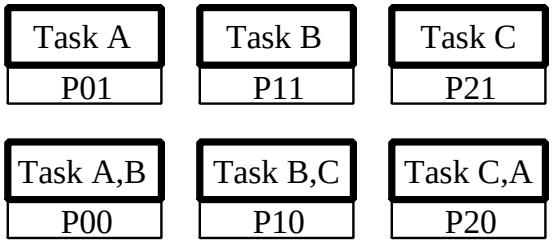
- Comparison targets
  - a) Duplicated task allocation



    - Each task can be performed by only two processor cores to compare their results

    → If a permanent fault occurs, the whole system gets failed immediately

  - b) Triplicated task allocation
    - The basic operation is the same as the proposed quadruplicated system



    - In the retry-and-decision phase, compose a TMR and identify the faulty core

    → If the second permanent fault occurs in the remaining pairs, the whole system gets failed

Comparison result (2)

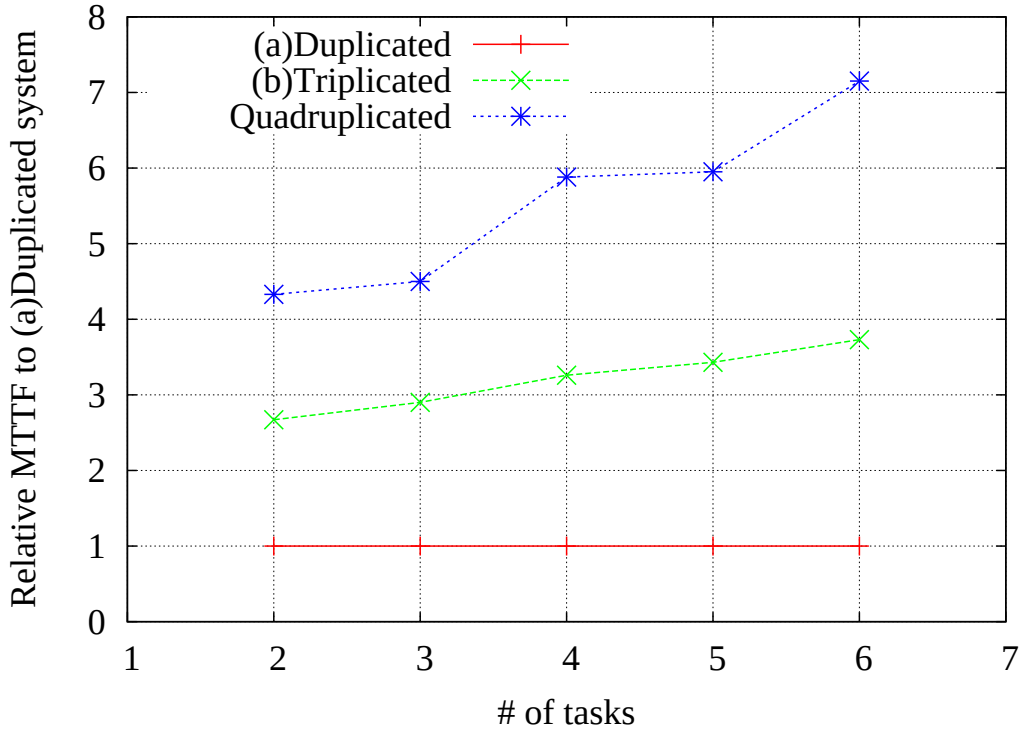
- No memory faults, 6 tasks
- The failure rate of a processor core: λ

Configuration	System MTTF
Duplicated	$0.0833/\lambda$ (1.00)
Triplicated (2-cyclic x3)	$0.311/\lambda$ (3.73)
Quadruplicated (2-cyclic x3)	$0.596/\lambda$ (7.15)

  - The MTTF of quadruplicated system is over 7 times longer than duplicated system while the amount of memory is only twice

Comparison result (2)

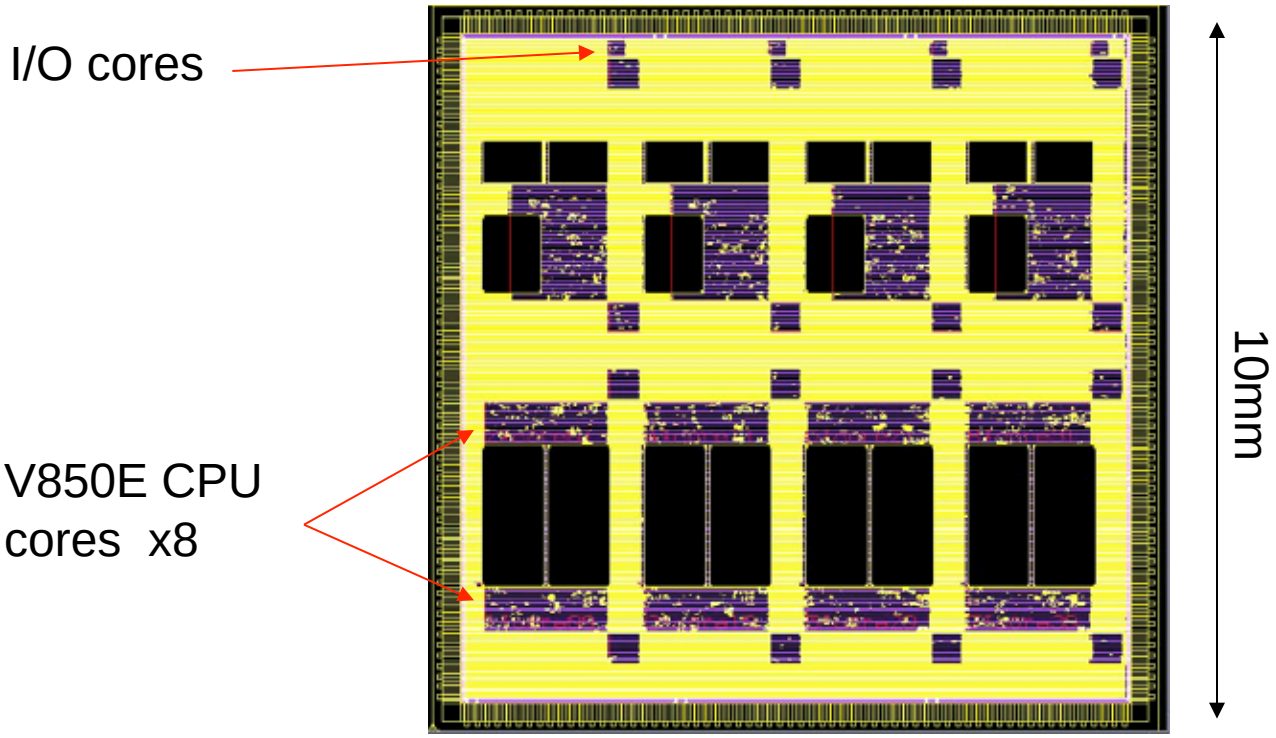
- Scalability
  - The ratio of MTTF to the duplicated configuration



  - The proposed quadruplicated static task allocation method is effective as the number of tasks increases

Chip implementation

- Using 130nm process technology



- We are now testing and evaluating this chip

Conclusion

- We have proposed a processor-level fault tolerance technique for NoC-based MPSoCs where each processor core has its small private memory
  - Each task is quadruplicated and statically assigned to private memories so that each memory has only two different tasks and fine-grain task pairs can be composed
- We have evaluated the MTTF of the proposed task allocation method
  - The MTTF is over 4.3 times longer than that of the duplicated task allocation
  - It is more effective as the number of simultaneously executed tasks increases