DAG Scheduling Algorithm Considering Large-scale Calculation Tasks Using Many-core Architecture

Yuto Kitagawa[1]
Takuya Azumi[2]

1 : Graduate School of Engineering Science, Osaka University, Japan
2 : Graduate School of Science and Engineering, Saitama University, Japan

Outline

- Introduction
- System Model
  - KALRAY MPPA-256
  - DAG notations
- Open problems and our approaches
  - Open problems
  - Scheduling assumptions
  - Proposed algorithm based on list-scheduling
    - Task-prioritizing phase
    - Processor-selection phase
  - Scheduling example
- Conclusions

[Introduction]

Multi/many core and embedded systems

- High computing performance and low power consumption are needed in embedded systems
  - **Examples:**
    - Autonomous driving system

- Many-core hardwares for embedded systems are suitable for large-scale and parallel computation

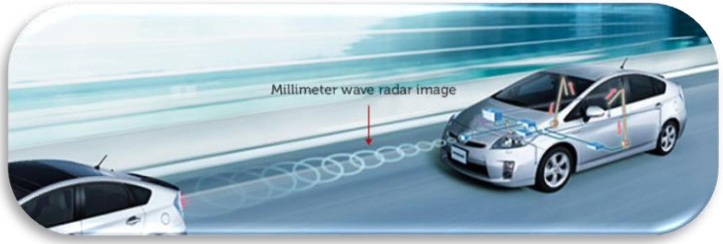
Kalray MPPA-256 (256 cores)

Tilela TILE-Gx100 (100 cores)

Deadlines for automotive systems

- Automotive systems require a strict real-time performance
  - Deadline miss leads to a fatal accident
  - Operation is statically determined at a development stage

- Multiple applications operate in automotive systems
  - There are multiple deadlines

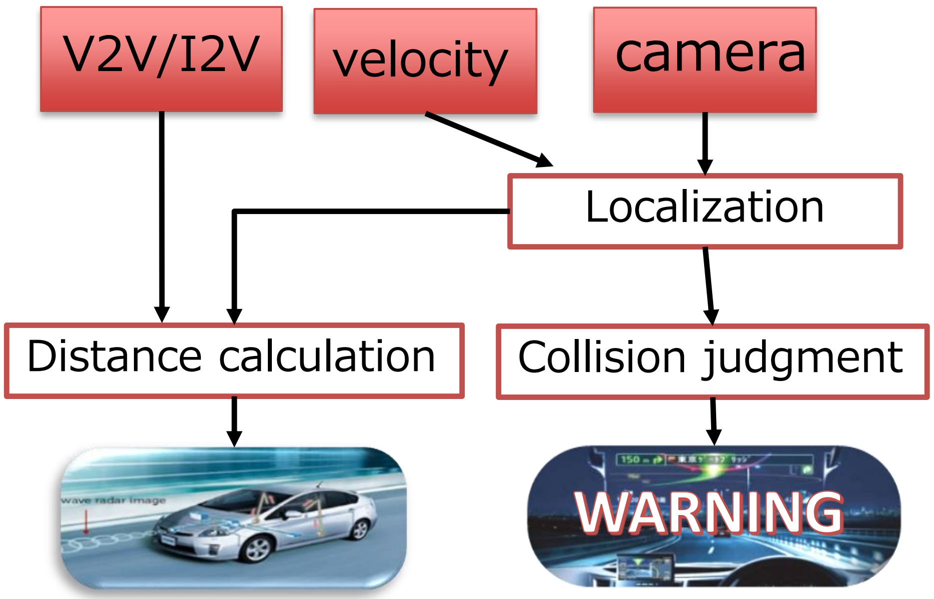


Automatic brake

Collision warning

Scheduling Problem with MPPA-256

- There are various processes up to applications
  - Applications with different deadlines are mixed
  - We propose a static non-preemptive scheduling method to meet deadlines with Kalray MPPA-256.



Deadline : 300 msec

Automatic brake

Deadline : 100 msec

Collision warning

Kalray MPPA-256

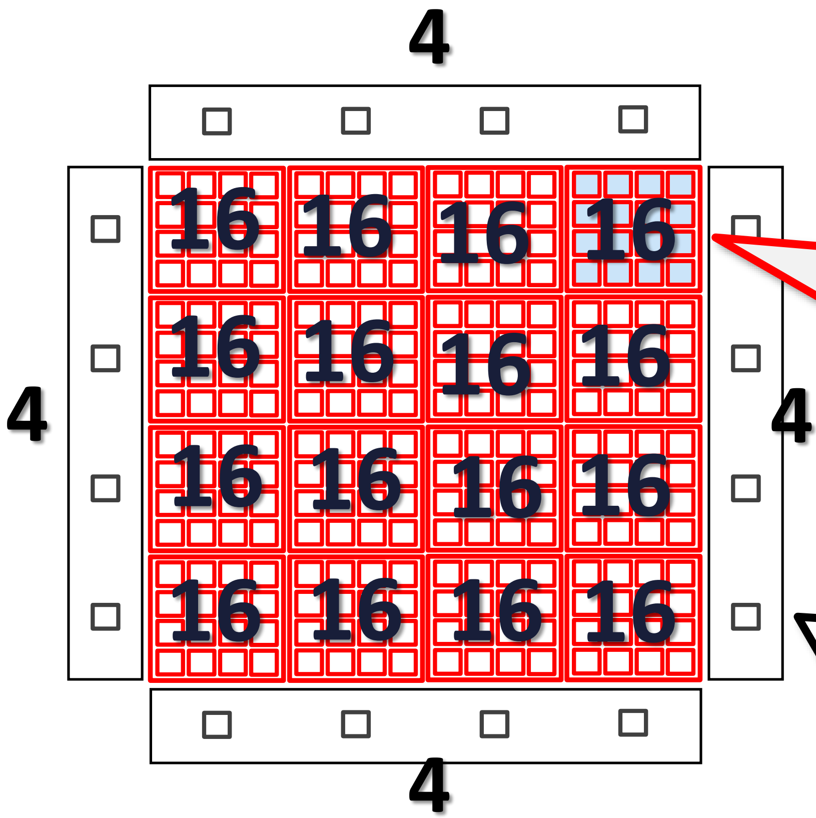
High computing performance

Lower power consumption

KALRAY MPPA-256

●KALRAY MPPA-256 architecture

- **Scalability: 256 + 16 cores**
- **High power efficiency**
- **NoC (Network-on-Chip)**



Compute Cluster (CCs)

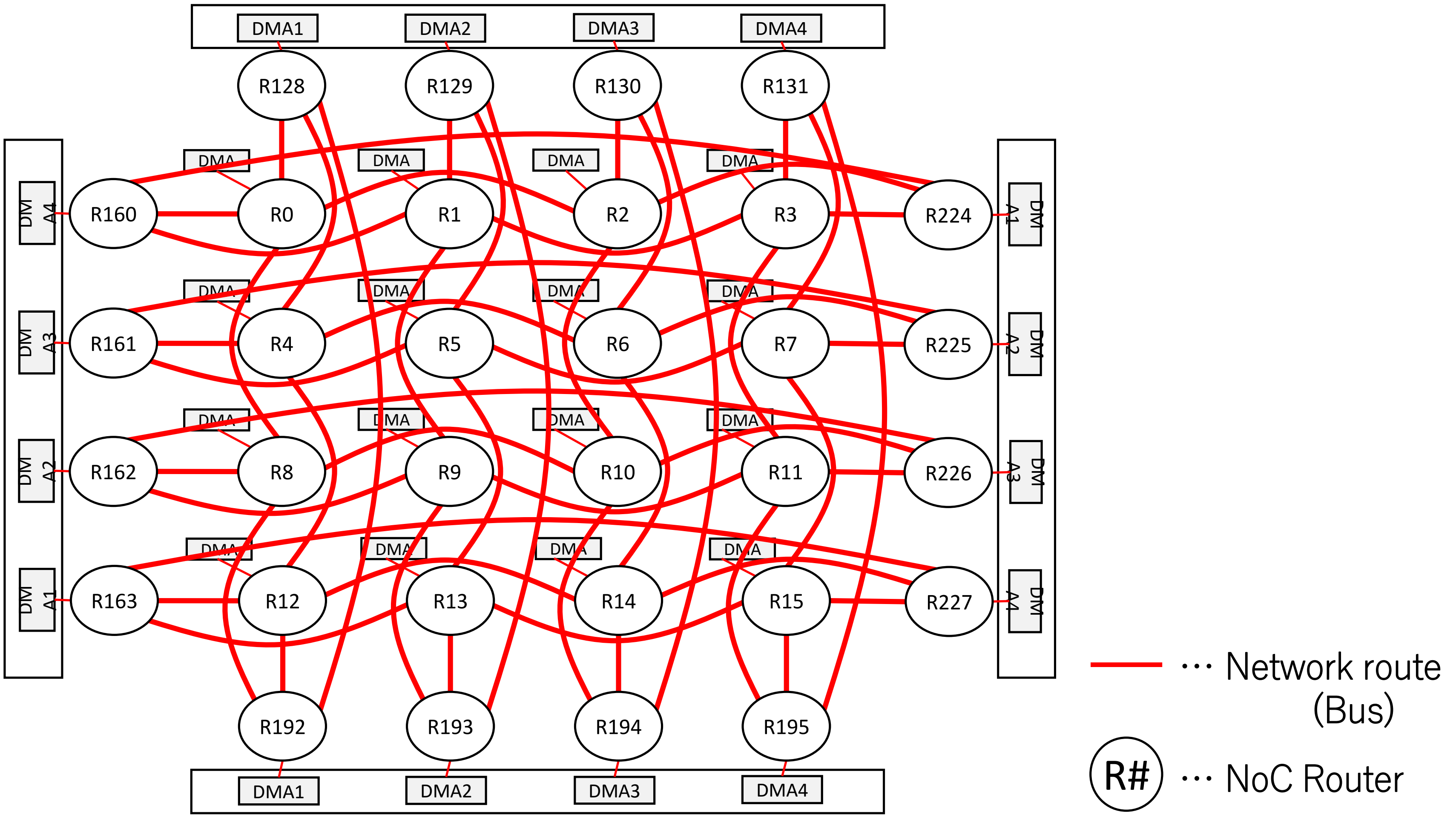
- **16 computing cores** per cluster
- NoC interface

I/O Cluster

- **4 IO cores** per cluster
- NoC interface
- Ethernet, PCIe

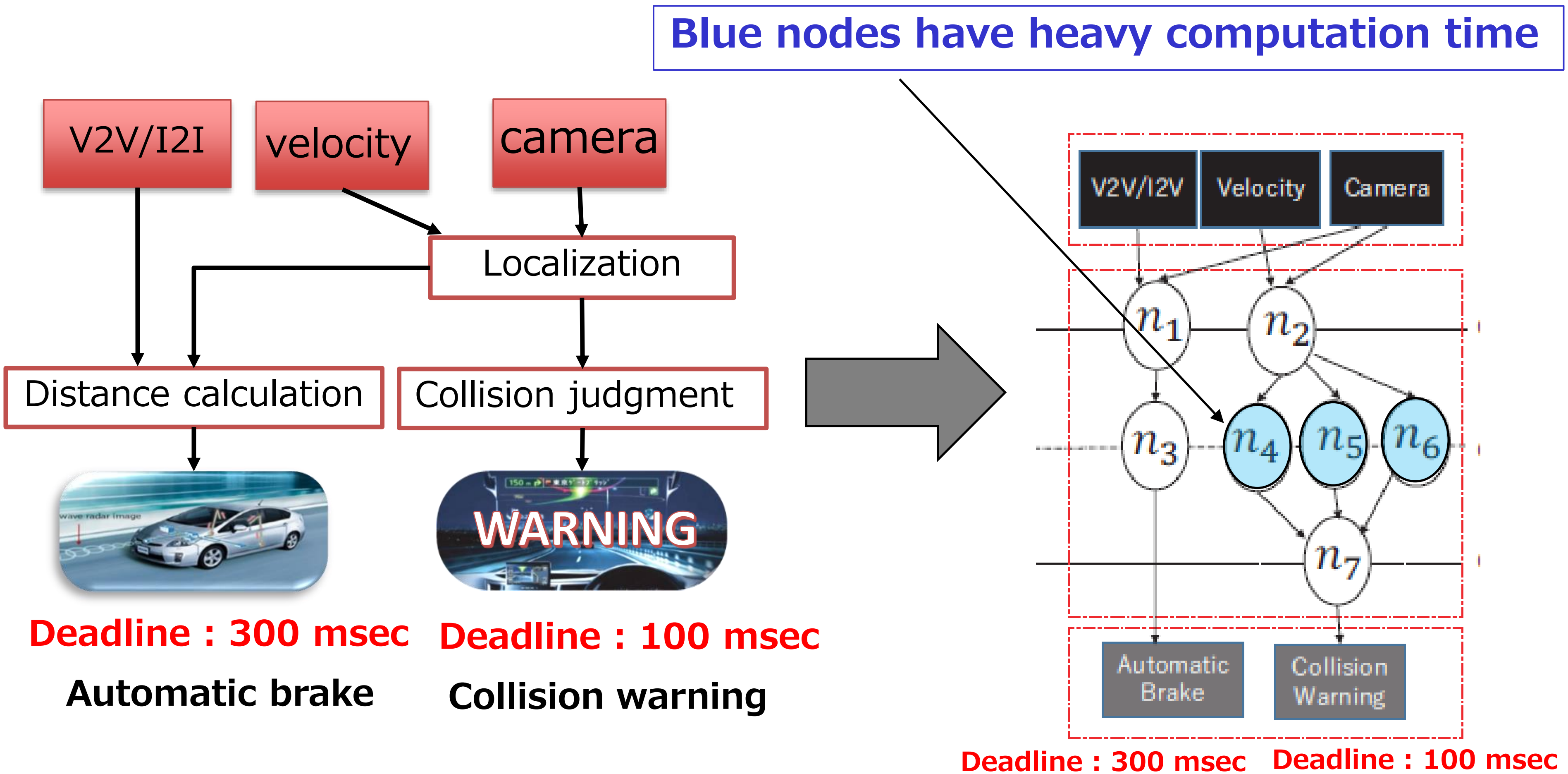
[System Model]
KALRAY MPPA-256/NoC Map

- Network-on-Chip (NoC)



DAG notations

- Data flow for an automotive system can be described as a Direct Acyclic Graph(DAG)



DAG notations

- **Graph $G =< V(G), E(G) >$**

$V(G) = \{n_1, n_2, \dots, n_{|V(G)|}\}$ Node set

$E(G) \subseteq V(G) \times V(G)$ Direct edge set

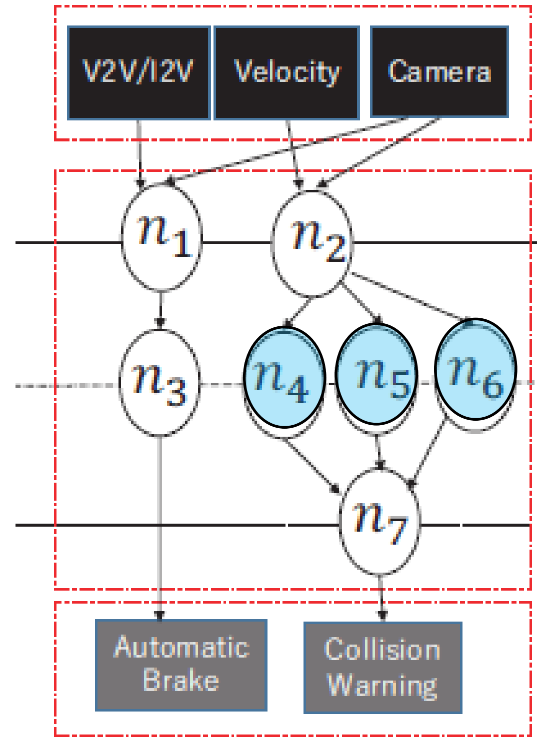
$comp(n_i)$: computation time of n_i

$pred(n_i)$: a set of n_i's immediate predecessor nodes

$succ(n_i)$: a set of n_i's immediate successor nodes

$depth(n_i)$: a length of the longest path from entry nodes to n_i.

$comm_{s,d}$: communication time from n_s to n_d



- $e_{s,d} \in E(G)$ indicate n_d can begin execution only after n_s completes transmission of a computation result

Open Problems

1. Our scheduling problem is an NP complete problem

2. We determine processing cores of KALRAY MPPA-256 for nodes with large computation time and those with not

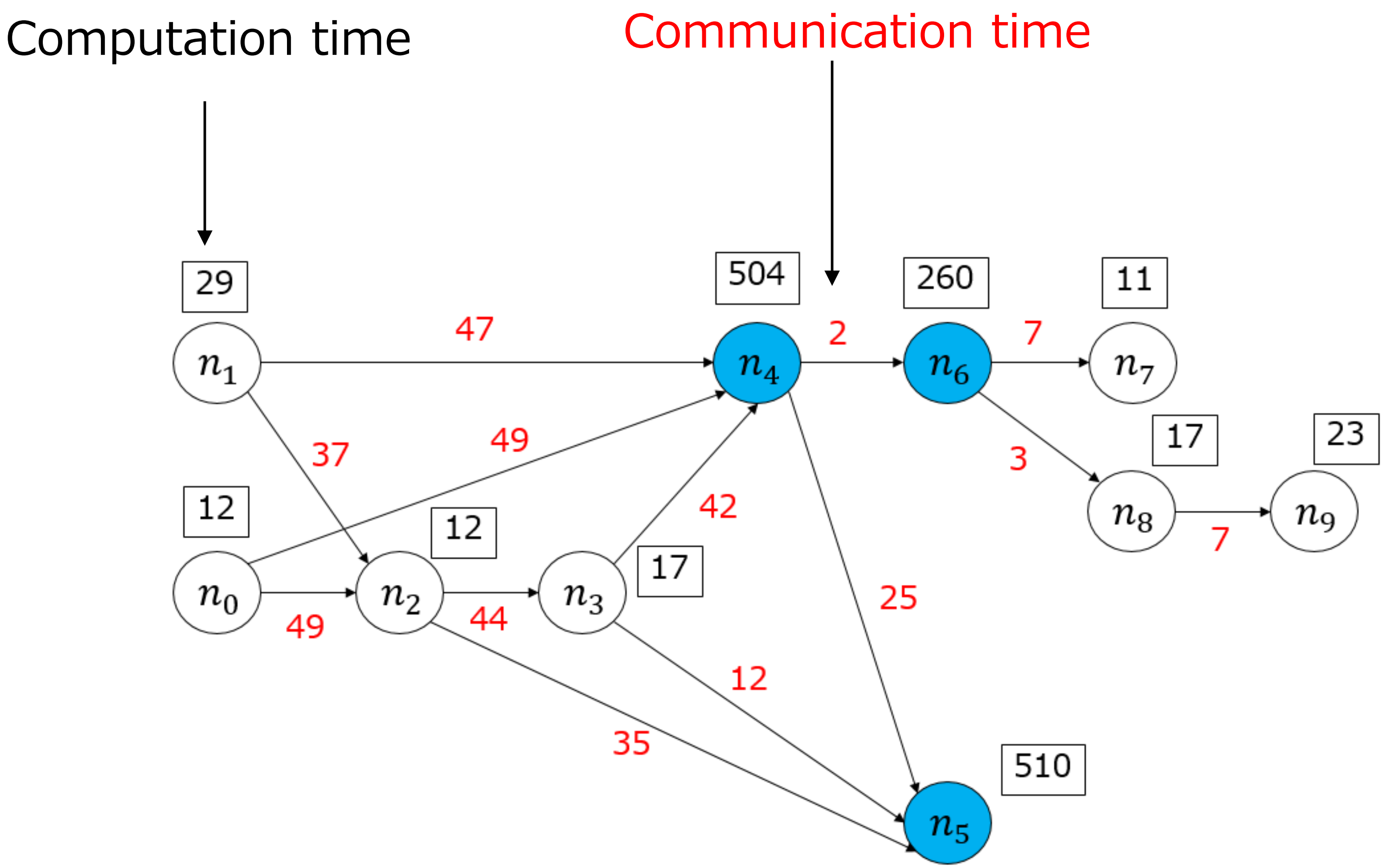
Our approaches to open problems

1. Our scheduling problem is NP complete
   - We use a list-scheduling method (Heuristic method)

2. We determine processing cores of KALRAY MPPA-256 for nodes with large computation time and those with not
   - We allocate cores using node’s calculation time and a position to a DAG

[Proposed scheduling algorithm]

Motivation example of a DAG

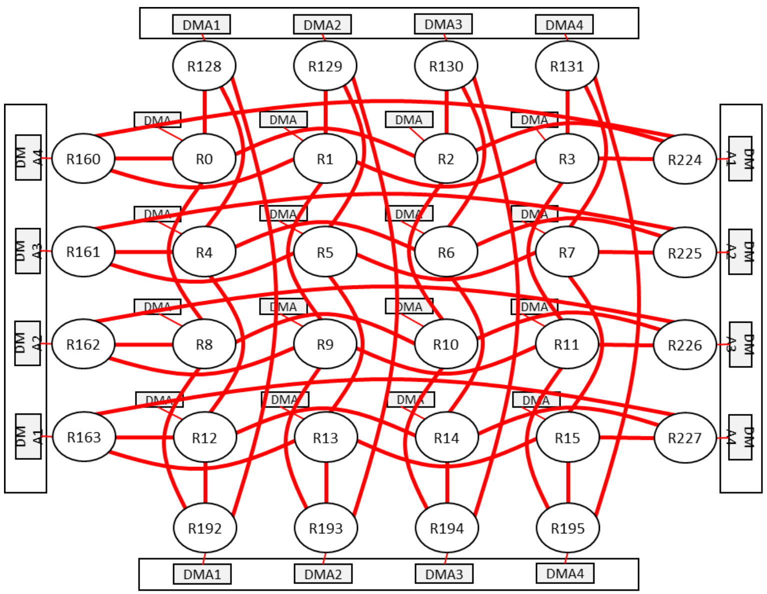


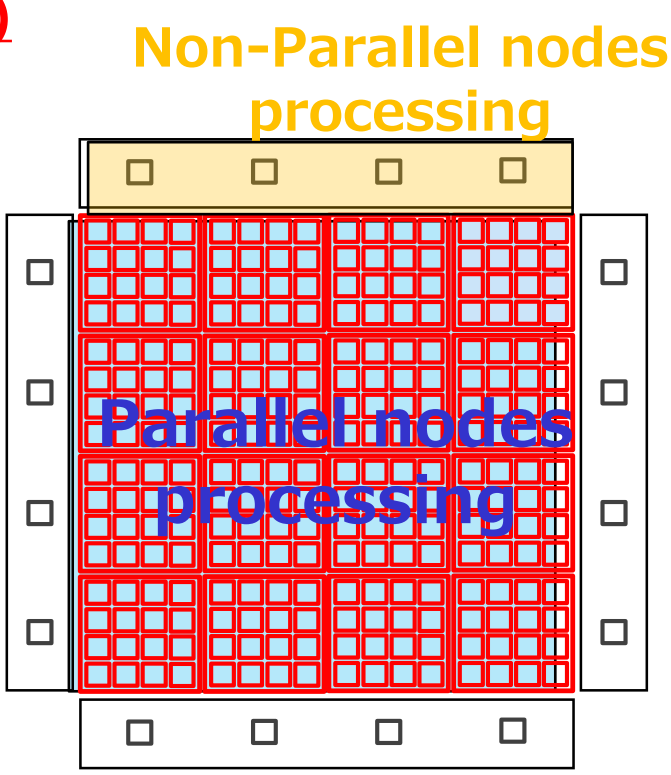
- We call a white node as a **non-parallel node** and call a blue node as a **parallel node**

Parallel computation are needed in parallel nodes

Scheduling assumptions

- **Processing core assignment**
  - **Non-parallel nodes → four IOS cores**
  - **Parallel nodes → 256 cores (16 CCs)**





- **The reason of core assignment**
  - We utilize CC resources for parallel computation

Scheduling assumptions

- Amdahl's law
  - performance improvement rate achieved when the degree of parallelism of the computer is increased

- Performance improvement rate : $S(N)$

$$S(N) = \frac{1}{(1-K) + \frac{K}{N}}$$

Non parallelizable　　parallelizable

K : the ratio of parallelizable part to entire execution time

N : the number of cores

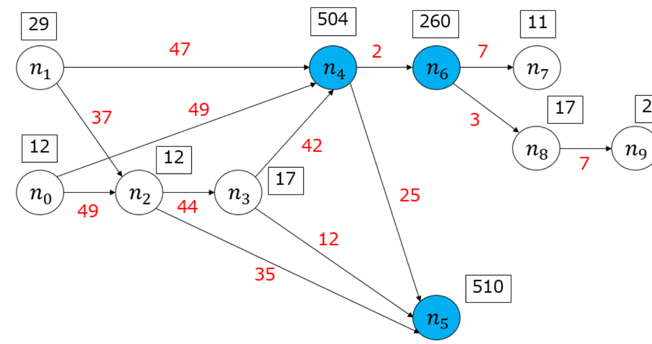
- Assumption : Non-parallel nodes → $\boldsymbol{K = 0}$
  Parallel nodes → $\boldsymbol{K \neq 0}$

Proposed many-core scheduling algorithm

- Based on **list-scheduling algorithm**
  - A heuristic method

- Scheduling process
  1. Priority is given to all nodes (Task-prioritizing phase)
  2. Processor selection is performed from a node with a high priority (Processor selection phase)

Task-prioritizing phase



●Step1 : Task-prioritizing phase

- We calculate a **rank** for each task
- **From exit nodes to entry nodes**

●Non-parallel nodes (for exit nodes)

$$rank_{non-parallel}(n_{exit}) = comp(n_{exit})$$

●Non-parallel nodes (for other nodes)

$$rank_{non-parallel}(n_s) = comp(n_s) + \max \left\{ \begin{array}{c} \max\limits_{n_d \epsilon succ(n_s)} \{rank_{non\_parallel}(n_d) + comm_{s,d}\}, \\ \max\limits_{n_d \epsilon succ(n_s)} \{rank_{parallel}(n_d) + comm_{s,d}\} \end{array} \right\}$$

Task-prioritizing phase

●Step1 : Task-prioritizing phase

●Parallel nodes (for exit nodes)

$$rank_{parallel}(n_{exit}) = \frac{comp(n_{exit})}{S(CC_{request})}$$

Amdahl's law

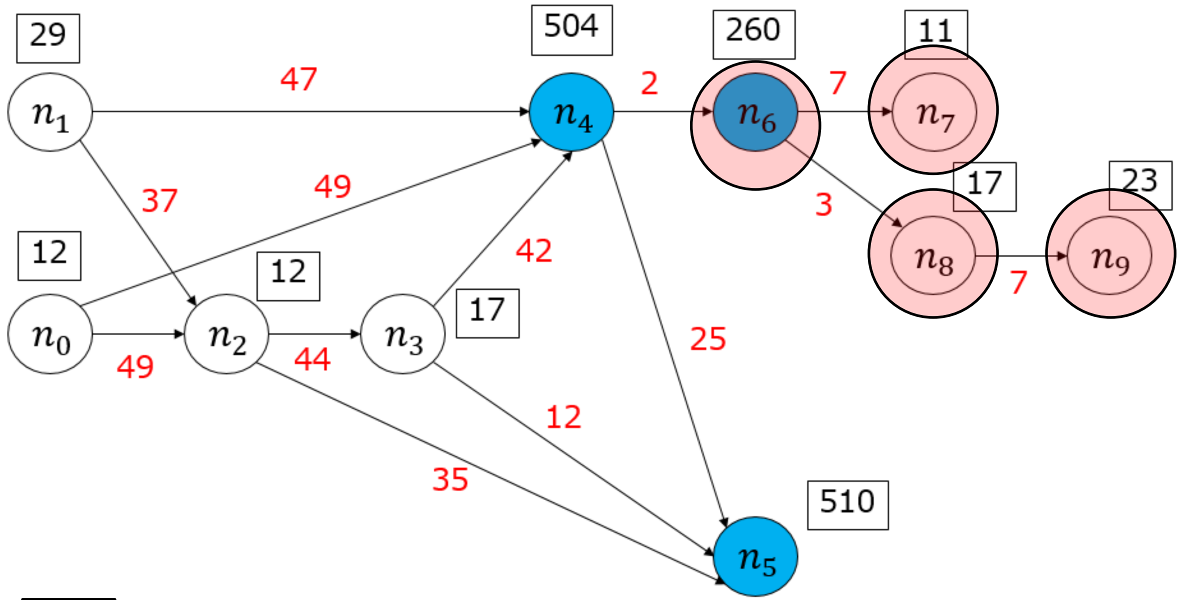
●Parallel nodes (for other nodes)

$$rank_{parallel}(n_s) = \frac{comp(n_s)}{S(CC_{request})} + \max\left\{\begin{array}{c} \max\limits_{n_d \epsilon succ(n_s)} \{rank_{non\_parallel}(n_d) + comm_{s,d}\}, \\ \max\limits_{n_d \epsilon succ(n_s)} \{rank_{parallel}(n_d) + comm_{s,d}\} \end{array}\right\}$$

Amdahl's law

Task-prioritizing phase

●An illustrative example(Task-prioritizing phase)



$rank(n_9) = 23$

$rank(n_8) = 17 + 23 + 7 = 47$

$rank(n_7) = 11$

$rank(n_6) = 80 + \max\{11 + 7, 47 + 3\} = 130$

Computation time

Successor rank + communication time

Task-prioritizing phase

●Calculate final rank

- Non-parallel nodes

$$final\ rank_{non-parallel}(n_i) = rank_{non-parallel}(n_i) * \frac{1}{depth(n_i)^2}$$

- Parallel nodes

$$final\ rank_{parallel}(n_i) = rank_{parallel}(n_i) * \frac{1}{depth(n_i)^2}$$

●Results

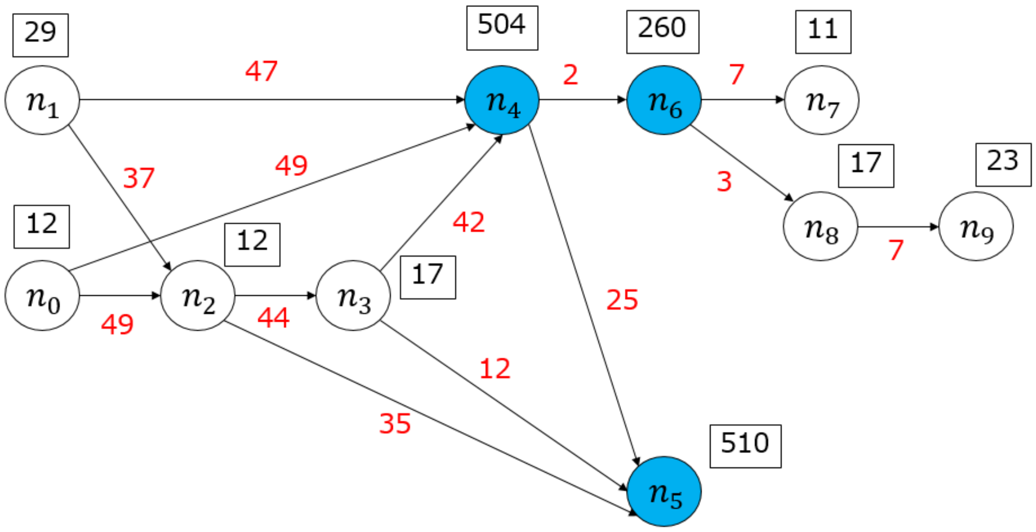
Node	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
$rank(n_i)$	508	513	447	391	332	155	130	11	47	23
$depth(n_i)$	1	1	2	3	4	5	5	6	6	7
$CC_{request}$	0	0	0	0	16	11	5	0	0	0
$final\ rank(n_i)$	508	513	111	43	20	6.2	5.21	0.31	1.31	0.47

Task-prioritizing phase

- Put nodes to a priority queue from large $final\,rank$ value

Priority Queue $= \{n_7, n_9, n_8, n_6, n_5, n_4, n_3, n_2, n_0, n_1\}$

Low priority **High priority**



Processor selection phase

- **Step2 : Processor selection phase**

- **Non-parallel nodes**
  - We decide **EST (Earliest Start Time)** and **EFT (Earliest Finish Time)**
  - For each processors, EST is determined by maximum values of
    - the target processor is available
    - processing end time of preceding nodes + communication time

$$EST_{non-parallel}(n_d, p_{dest}) = \max\{avail[p_{dest}], \max_{\substack{n_s \epsilon pred(n_d) \\ proc(n_s)=p_s}}\{EFT_{non-parallel}(n_s, p_s) + comm_{s,d}\}, \max_{\substack{n_s \epsilon pred(n_d) \\ proc(n_s)=CC_{request}}}\{EFT_{parallel}(n_s, CC_{request}) + comm_{s,d}\}\}$$

Processor selection phase (non-parallel nodes)

- **Non-parallel nodes**
  - EFT is determined by

$$EFT_{non-parallel}(n_d, p_{dest}) = EST_{non-parallel}(n_d, p_{dest}) + comp(n_d, p_{dest})$$

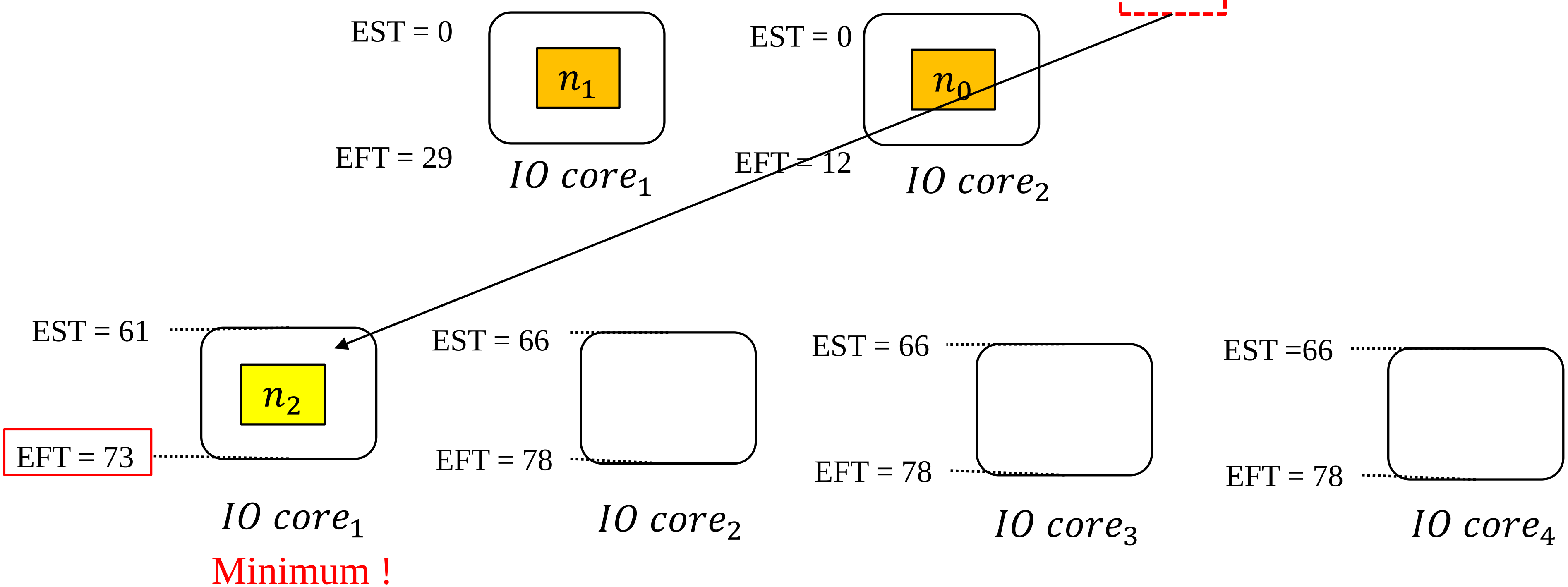
- We allocate a non-parallel node to a processor that yields the smallest EFT.

Processor selection phase (non-parallel nodes)

- **An illustrative example (processor selection of non-parallel nodes)**

Low priority High priority

Priority Queue $=\{n_7, n_9, n_8, n_6, n_5, n_4, n_3, n_2, n_0, n_1\}$



Processor selection phase (parallel nodes)

- **Parallel nodes**
  - We utilize calculation time and a position to a DAG

16 process elements in one CC

$$\boldsymbol{CC_{request}(n_i)} = \left(\boldsymbol{16} * \frac{\boldsymbol{comp(n_i)}}{\sum_{n_k \epsilon D_{n_i}} \boldsymbol{comp(n_k)}}\right) * \boldsymbol{16}$$

D_{n_i} : A set of parallel nodes having the same depth value

- We determine EST and EFT of parallel nodes

$$EST_{parallel}(n_{entry}, p_{dest}) = \max\{avail[p_{dest}]$$

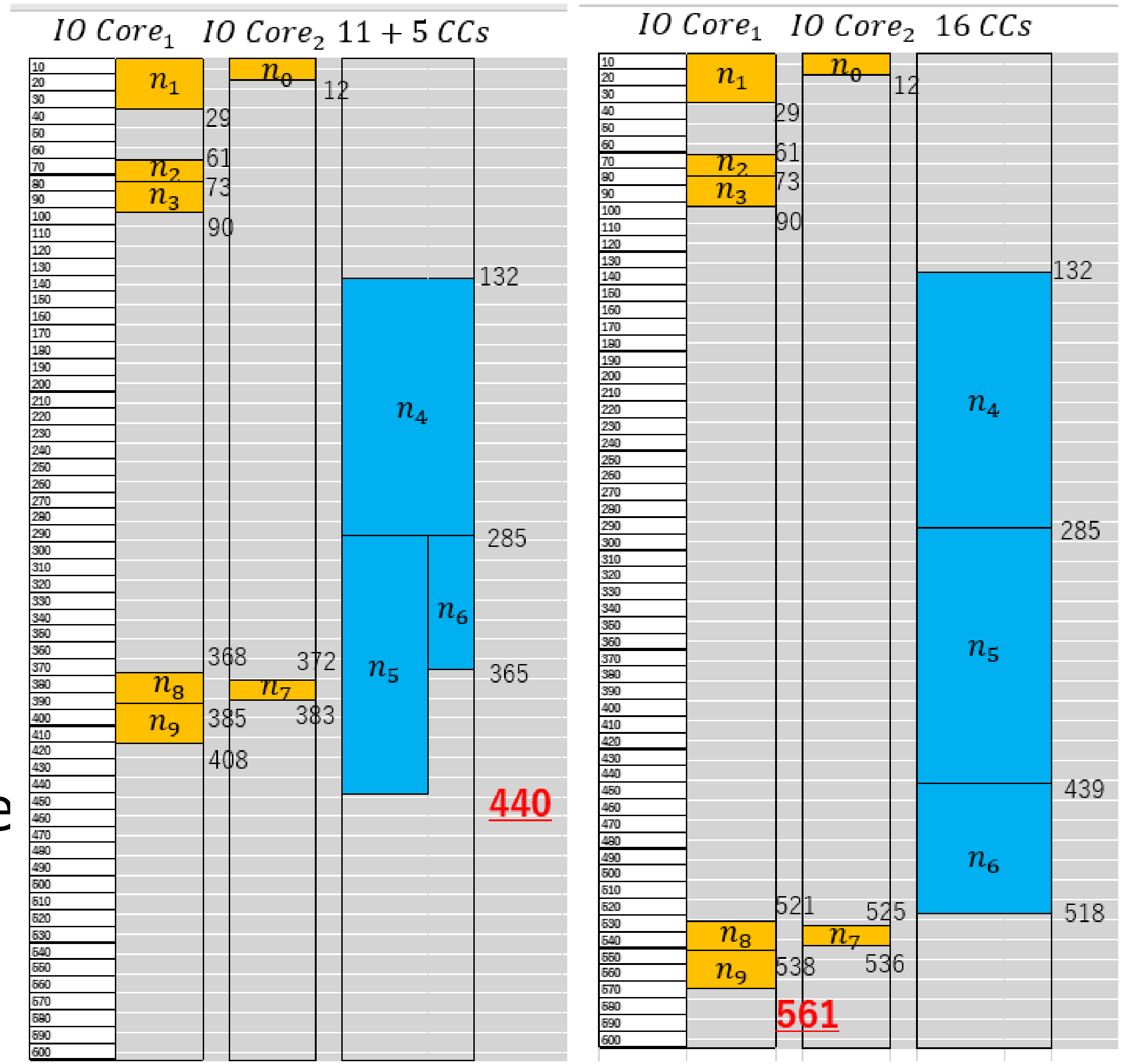
$$EST_{parallel}(n_d, p_{dest}) = \max\{avail[CC_{request}], \max_{\substack{n_s \epsilon pred(n_d) \\ proc(n_s)=p_s}}\{EFT_{non-parallel}(n_s, p_s), comm_{s,d}\}, \max_{\substack{n_s \epsilon pred(n_d) \\ proc(n_s)=CC_{request}}}\{EFT_{parallel}(n_s, CC_{request}), comm_{s,d}\}\}$$

$$EFT_{parallel}(n_d, p_{dest}) = EST_{parallel}(n_d, CC_{request}) + \frac{comp(n_d, p_{dest})}{CC_{request}}$$

[Proposed scheduling algorithm]

A scheduling example

- K=0.7 for all parallel nodes
- Left figure
  - We divide CCs using $CC_{request}$ equation
- Right figure
  - We do not divide CCs



Conclusions and future work

- Problems
  - Our scheduling problem is a NP complete problem
  - KALRAY MPPA-256 core allocation
- Approaches
  - Based on List-scheduling algorithm
  - Non-parallel node processing using four IOS cores
  - Parallel node processing using $CC_{request}$ equation
- Future work
  - We consider deadline in rank formula
  - We consider NoC link communication contention
  - We consider a pipeline scheduling
- **Discussion topic**
  - **Heuristic method**
  - **How to divide computer clusters (CCs)**