Master's Thesis

Scalable Parallel Computing
on NoC-based Embedded Many-Core Platform

Supervisor

Professor Toshimitsu Ushio

Assistant Professor Takuya Azumi

By

Yuya Maruyama

February xx, 2018

Division of Mathmatical Science for Social System,

Department of Systems Innovation,
Graduate School of Engineering Science, Osaka University

Abstract

This paper presents evaluations of parallel computing on embedded many-core platforms. In embedded systems, high processing requirements and low power consumption require heterogeneous computing platforms including many-core platforms. Considering nonuniform memory access (NUMA) for scalability, many-core applications must be designed based on scalable data allocation and scalable parallel computing. As a reference embedded many-core computing platform, we use the Massively Parallel Processor Arrays 256 (MPPA-256) developed by Kalray, one of the commercial off-the-shelf (COTS) multi-/many-core platforms. In this paper, we conduct evaluations of data transfer on network-on-chip (NoC) implementations, microbenchmarks, and parallelization of a practical application on NoC-based embedded many-core platforms. We investigate currently achievable data transfer latencies between distributed memories on NoC systems, memory access characteristics with NUMA, and parallelization scalability of many cores. As a practical application, we parallelize a portion of the self-driving algorithms on many-core processors. We believe that this performance level is acceptable for the real-world production of autonomous vehicles. By highlighting many-core computing capabilities, we explore scalable parallel computing on NoC-based embedded many-core platforms.

Contents

Abstract **i**

1 Introduction **1**

2 System Model **3**
2.0.1 Hardware Model 3
2.0.2 Software Model 6

3 Evaluations **11**
3.0.1 D-NoC Data Transfer 11
3.0.2 Matrix Calculation 14
3.0.3 Practical Application 17
3.0.4 Lessons Learned 18

4 Related Work **24**

5 Conclusions **28**

References **30**

Published List **33**

Chapter 1

Introduction

The evolution of a next-generation computing platform oriented toward having multi-/many-cores is necessary to satisfy the demand for increasing computation in conjunction with reasonable power consumption in several domains such as automobiles. This trend is also adapted to embedded systems because of high processing requirements. For example, self-driving systems involve various applications and are sometimes characterized by demands for high-performance computing. Considering the requirements of high processing, predictability, and energy efficiency for self-driving systems, such systems require a heterogeneous computing system such as multi-/many-core platforms and graphical processing units (GPU). (This is discussed in Section 4.) In embedded systems, multi-/many-core architectures are an important trend as the architecture integrates cores to realize high-performance and general-purpose computing with low power consumption [1], [2]. Extant studies have examined several applications of multi/many-core platforms [1], [3], [4], [2], [5] [6].

Based on the above background, multi/many-core platforms are fabricated and released as commercial off-the-shelf (COTS) multicore components. (e.g., the Massively Parallel Processor Arrays (MPPA) 256 developed by Kalray [7], Tile-Gx developed by Tilera [8] [9], Tile-64 developed by Tilera [10], Xeon Phi developed by Intel [11], [12], and Single-chip Cloud Computer (SCC) developed by Intel [13]) Several platforms such as MPPA-256 and Tile-64 target embedded systems, and the research focused on multi-/many-core platforms has received increasing attention [14], [15], [16]. We discuss comparisons of many-core platforms in Section 4.

In embedded many-core platforms, nonuniform memory access (NUMA) and distributed memories connected with network-on-chip (NoC) components are an important approach for core scalability and power consumption. Several COTS platforms include NoC technology and a cluster of many-core architectures in which cores are allocated closely. For instance, MPPA-256 and Tile-Gx72 have NoC components to share distributed memories instead of shared buses. Furthermore, MPPA-256 contains 16 clusters in which 16 cores are included and contains 256 general-purpose cores in total. Although the cores do not guarantee cache coherency, this significantly exceeds the number of cores in other COTS such as Tile-64 and Tile-Gx72. The clusters of cores are capable of running separate independent applications with respect to the desired power envelope of embedded

applications.

Despite the appearance of embedded many-core platforms, several difficulties persist in the adaptation of these platforms to embedded systems [1], [3]. In NoC-based embedded many-core platforms, data transfer between distributed memories with NoC technology, memory access characteristics in NUMA, and parallelization potential in practical applications are not fully unrevealed for application developers. We focus on the above issues and conduct evaluations to reveal the practicality of NoC-based many-core platforms. This work explores scalable parallel computing and data allocation with quantitative evaluations and a practical application. As a reference embedded many-core platform, we use MPPA-256 [7] which adopts NUMA using NoC technology and realizes numerous cores with low power consumption.

Contributions: This work focuses on examining embedded many-core computing based on NoC technology, such as MPPA-256. We conduct evaluations of data transfer methods on NoC components and microbenchmarks with matrix calculation to clarify latency characteristics of data transfer, parallel computing, and the influence of data allocation. Subsequently, we parallelize a localization algorithm that is the core of the self-driving system. We reveal the advantages and disadvantages of embedded many-core computing based on NoC technology in a quantitative manner by introducing the following contributions:

- The evaluations of the data transfer on DMA-capable NoC components quantitatively characterize the end-to-end latencies, which depend on the routing and DMA configurations.
- The scalability of parallelization in many-core processors based on NoC technology is observed in evaluations of matrix calculations in several situations and a real complex application as a part of a self-driving system.
- The evaluations of parallel processing examine the characteristics of the memory access speed, which varies according to where data are allocated and what accesses the memory.

To the best of our knowledge, this is the first work that examines data transfer and data allocation matters for many-core computing beyond an intuitive expectation to allow system designers to choose appropriate data transfer methods. Additionally, the speed-up result of a self-driving application indicates a practical potential for NoC-based embedded many-core computing.

Organization: The remainder of this work is organized as follows. The system model considered in this work is discussed in Section 2 in which the hardware model, namely Kalray MPPA-256 Bostan, and the system model are presented. Section 3 explains evaluation setup and approach, and illustrates experimental evaluations. Section 4 examines the related work that focuses on multi-/many-core systems. Section 5 presents the conclusions and directions for future research.

Chapter 2

System Model

This section presents the system model used throughout the work. The many-core model of Kalray MPPA-256 Bostan is considered. First, a hardware model is introduced in Section 2.0.1, followed by a software model in Section 2.0.2.

2.0.1 Hardware Model

The MPPA-256 processor is based on an array of computing clusters (CCs) and I/O subsystems (IOSs) that are connected to NoC nodes with a toroidal two-dimensional topology (as shown in Figs. 2.1, 2.2 and 2.3). The MPPA many-core chip integrates 16 CCs and 4 IOSs on NoC. The architecture of Kalray MPPA-256 is presented in this section.

I/O Subsystems (IOS)
MPPA-256 contains the following four IOSs: North, South, East, and West. The North and South IOSs are connected to a DDR interface and an eight-lane PCIe controller. The East and West IOSs are connected to a quad 10 Gb/s Ethernet controller. Two pairs of IOSs organize two I/O clusters (IOCs), as shown in Fig. 2.1. Each IOS consists of quad IO cores and a NoC interface.

- **IO Cores**: IO cores are connected to a 16-bank parallel shared memory with a total capacity (IO SMEM) of 2 MB, as shown in Fig. 2.1. The four IO cores have their own instruction cache 8-way associative corresponding to 32 (8×4) KB and share a data cache 8-way with 128 KB and external DDR access. The sharing of the data cache of 128 KB allows coherency between the IO cores. Additionally, IO cores operate controllers for PCIe, Ethernet, and other I/O devices. They operate the local peripherals, including NoC interfaces with DMA. It is also possible to conduct an application run on the IO cores.
- **NoC Interface**: The NoC interface contains four DMA engines (DMA1-4) and four NoC routers, as shown in Fig. 2.1, and the IOS DMA engine manages transfers between the IO SMEM, IOS DDR, and IOS peripherals (e.g., PCIe interface and Ethernet controllers). The DMA engine transfers data between routers via NoC through NoC

routers and has the following three NoC interfaces: a receive (Rx) interface, a transmit (Tx) interface, and a microcore (UC). The UC is a fine-grained multithreaded engine that can be programmed to set threads sending data with a Tx interface. The UC can extract data from memory by using a programmed pattern and send the data on the NoC. After the UC is initiated, this continues in an autonomous fashion without using a processing element (PE) and an IO core.

Compute Clusters (CCs)
In MPPA-256, the 16 inner nodes of the NoC correspond to the CCs. Fig. 2.2 illustrates the architecture of each CC.

- **PEs and an RM**: In a CC, 16 PEs and an RM share 2 MB cluster local memory (SMEM), so that 17 k1-cores (a PE or the RM) share 2 MB SMEM. Users use the PEs primarily for parallel processing. Developers spawn computing threads on PEs. The PEs and an RM in the CC correspond to the Kalray-1 cores, which implement a 32-bit 5-issue Very Long Instruction Word architecture with 600 or 800 MHz. Each core is fitted with its own instruction and data caches. Each cache is 2-way associative with a capacity of 8 KB. Note that these caches do not guarantee cache coherency.
- **A Debug Support Unit and a NoC Interface**: In addition to PEs and an RM, bus masters on the SMEM correspond to a Debug Support Unit and a DMA engine in a NoC interface. A DMA engine and a NoC router are laid out in a NoC interface. The CC DMA engine also has the following three interfaces: an Rx, a Tx, and a UC. The CC DMA engine is instantiated in every cluster and connected to the SMEM.

Network-on-Chip (NoC)
The 16 CCs and the four IOSs are connected by NoC as shown in Fig. 2.3. Furthermore, NoC is constructed as the bus network and has routers on each node.

- **Bus Network**: A bus network connects nodes (CCs and IOSs) with torus topology

[17], which involves a low average number of hops when compared to mesh topology [18], [19]. The network is actually composed of the following two parallel NoCs with bidirectional links (denoted by red lines in Fig. 2.3): the data NoC (D-NoC) that is optimized for bulk data transfers and the control NoC (C-NoC) that is optimized for small messages at low latency. The NoC is implemented with wormhole switching and source routing. Data is packaged in variable length packets that are broken into small pieces called flits (flow control digits). The NoC traffic is segmented into packets, with each packet including 1-4 header flits and 0-62 payload data flits.

- **NoC routers**: One node per CC and four nodes per I/O subsystem hold the following two routers of their own: a D-NoC router and a C-NoC router. Each RM or IO core on a NoC node is associated with the two aforementioned NoC routers. Furthermore, DMA engines in a NoC interface on the CC/IOS send and receive flits through the D-

NoC routers with the Rx interface, the Tx interface, and the UC. A mailbox component corresponds to the virtual interface for the C-NoC and enables one-to-one, N-to-one, or one-to-N low-latency synchronization. The NoC routers shown in Fig. 2.2 illustrate nodes as R1-16, R128-131, R160-163, R224-227, and R192-195 in Fig. 2.3. For purposes of simplicity, D-NoC/C-NoC routers are illustrated with a NoC router. In both D-NoC and C-NoC, each network node (a CC or an IOS) includes the following 5-link NoC routers: four duplexed links for North/East/West and South neighbors and a duplexed link for local address space attached to the NoC router. The NoC routers include FIFOs queuing flits for each direction. The data links are four bytes wide in each direction and operate at the CPU clock rates of 600 MHz or 800 MHz, with the result that each tile can transmit/receive a total of 2.4 GB/s or 3.2 GB/s, which is spread across the four directions (i.e., North, South, East, and West).

2.0.2 Software Model

The software stack used for Kalray MPPA-256 is composed of a hardware abstraction layer, a low-level layer, an OS, and a user application. Fig. 2.4 shows the software stack used for Kalray MPPA-256 in the present work. The Kalray system is an extensible and scalable array of computing cores and memory. With respect to the scalable computing array of the system, it is possible to map several programming models or runtimes such as Linux, a real-time operating system, POSIX API, OpenCL, and OpenMP. Each layer is described in detail.

In the hardware abstraction layer, an abstraction package abstracts hardware of a CC,

an IOS, and NoC. The hardware abstraction is responsible for partitioning hardware resources and controlling access to the resources from the user-space operating system libraries. The abstraction sets-up and controls interpartition communications as a virtual machine abstraction layer. The hardware abstraction runs on the dedicated RM core. All the services are provided commonly by an operating system (e.g., virtual memory and schedule) that must be provided by user-space libraries. A minimal kernel avoids wastage of resources and mismatched needs.

In a low-level library layer, the Kalray system also provides Libnoc and Librouting for handling NoC. Additionally, NoC features such as routing and quality of service are set by the programmer. Libnoc allows direct access to memory mapped registers for their configurations and uses. This library is designed to cause minimum CPU overhead and also serves as a minimal abstraction for resource allocation. Librouting offers a minimal set of functions that can be used to route data between any clusters of the MPPA. Routing on the network is conducted statically with its own policy. In addition, Libpower enables spawning and waiting for the end of execution of other clusters.

Several operating systems support the abstraction package in the OS layer. It is difficult for numerous cores such as MPPA to support previous operating systems for single/multicore(s) owing to problems involving parallelism and cache coherency [20], [21]. Here, the following real-time operating systems (RTOSs) supporting MPPA are introduced:

- **RTEMS**: Real-Time Executive for Multiprocessor Systems (RTEMS) is a full featured RTOS prepared for embedded platforms. RTEMS supports several APIs and standards, and most notably supports the POSIX API. The system provides a rich set of features, and an RTEMS application is mostly a regular C or C++ program that uses the POSIX API. RTEMS runs on the IOC except for the CC.
- **NodeOS**: On the CC, the MPPA cluster operating system utilizes a runtime called NodeOS. The OS addresses the need for a multicore OS to conform to the maximum possible extent to the standard POSIX API. The NodeOS enables a user code by using the POSIX API to run on PEs on the CC. First, NodeOS runtime starts on PE0 prior to calling the user main function. Subsequently, `pthread` is called on other PEs.
- **eMCOS**: On both CCs and IOSs, eMCOS provides minimal programming interfaces and libraries. Specifically, eMCOS is a real-time embedded operating system developed by eSOL (a Japanese supplier for RTOSs) and is the first commercially available many-core RTOS for use in embedded systems. The OS implements a distributed microkernel architecture. This compact microkernel is equipped with only minimal functions. The eMCOS enables applications to operate priority based message passing, local thread scheduling, and thread management on IOSs as well as CCs.

RTMES and NodeOS are provided by Kalray and eMCOS is released by eSOL.

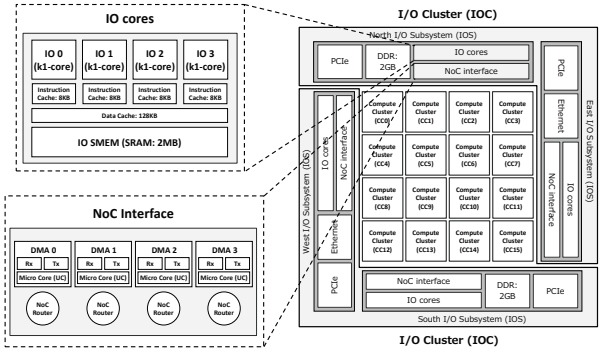

Fig. 2.1 An overview of the architecture of the Kalray MPPA-256 Bostan.

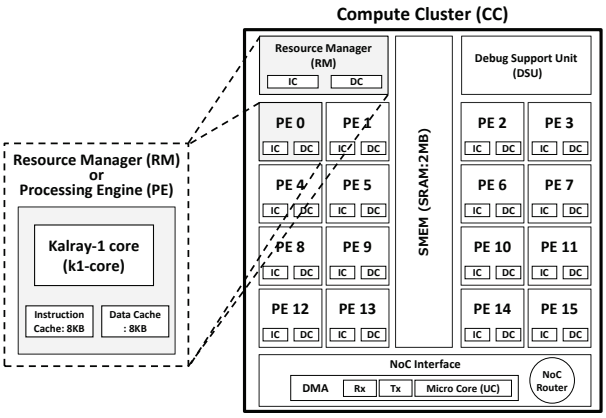

Fig. 2.2 Computing Cluster architecture.

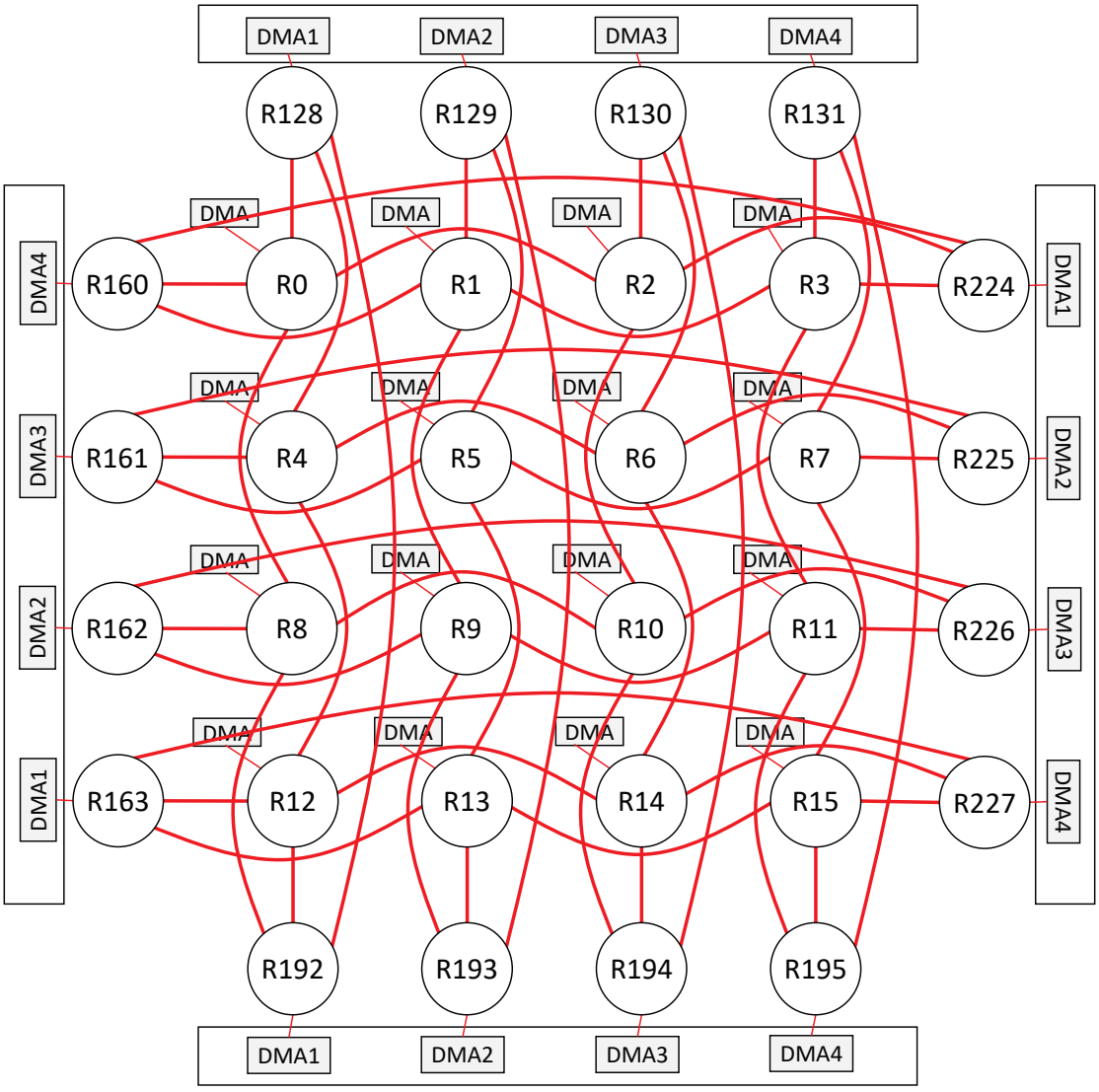

Fig. 2.3 NoC connections (both D-NoC and C-NoC).

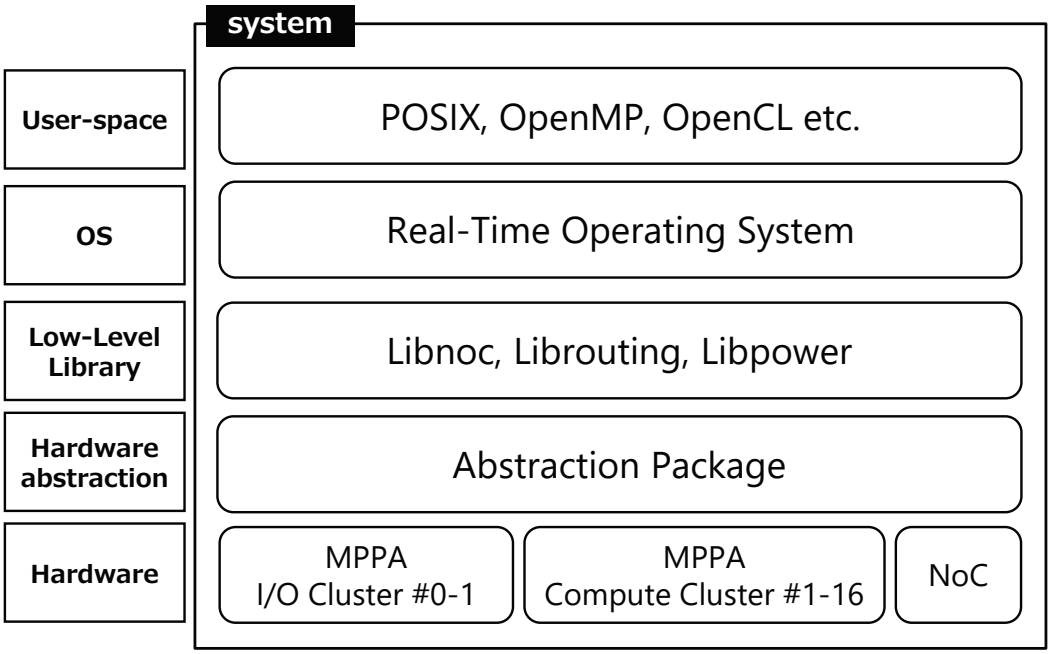

Fig. 2.4 The software stack of Kalray MPPA-256.

Chapter 3

Evaluations

First, this section involves examining two types of evaluations: a D-NoC data transfer evaluation in which latency characteristics of interfaces and memory type are explored and a matrix calculation evaluation that demonstrates the parallelization potential of the MPPA-256 and its memory access characteristics. Subsequently, we conduct a practical self-driving application to examine the practicality of NUMA many cores. The following evaluations are all conducted on real hardware boards with eMCOS.

3.0.1 D-NoC Data Transfer

Situations and Assumptions
This evaluation involves clarifying end-to-end latency by considering the relation among interfaces (Tx or UC), routing on NoC, and memory type (DDR or SMEM). This is achieved by preparing four routes as shown in Fig. 3.1. The routes on the D-NoC map (Fig. 2.3) contain various connections between routers, a direct link, a cross-link, and a flying link. With respect to the case of routes from the IOS routers to the CC routers, transmitted data are allocated in the DDR or IO SMEM. The CC includes only the SMEM as shown in Fig. 2.2. The transferred data correspond to 100 B, 1 KB, 10 KB, 100 KB, and 1 MB. The buffers are sequentially allocated in DDR or SRAM (IO SMEM or CC SMEM). The capacity of the CC SMEM is 2 MB, and thus it is assumed that the appropriate communication buffer size is 1 MB. Given this assumption, the other memory area corresponds to the application, libraries, and an operating system. End-to-end latencies are measured 1,000 times in numerous situations as shown in Figs. 3.2, 3.3, and 3.4, and boxplots are obtained, as depicted in Figs. 3.5, 3.6, and 3.7. In the evaluation setting, we minimize traffic conflicts to focus on the relation between end-to-end latency and routing on NoC.

Data Transfer Methods
In this section, data transfer methods in MPPA-256 are explained. For scalability purposes, MPPA-256 accepts a clustered architecture that avoids frequent memory contention between numerous cores, and it also accepts clustered architectures in which each cluster contains its own memory. Sixteen cores are packed as a cluster sharing 2 MB memory

(SMEM), as shown in Fig. 2.2. This avoids frequent memory contention due to having numerous cores and helps in increasing the number of cores. However, the architecture constrains memory that can be directly accessed by the cores. Communicating with cores outside the cluster requires transferrinng data between clusters through the D-NoC with NoC interfaces.

An Rx interface exists on the receiving side to receive data with DMA. It is necessary to allocate a D-NoC Rx resource and configure the Rx to wait to receive the data. A DMA in a NoC interface contains 256 D-NoC Rx resources. Two interfaces, the Tx interface and the UC interface as explained in Sections 2.0.1 and 2.0.1, are present with respect to the sending side for users to send data between clusters. The UC is a network processor

that is programmed to set threads to send data in DMA. The UC executes programmed patterns and sends data through the D-NoC without a PE or an RM. The UC interface results in higher data transfer throughput compared to the direct activation of the Tx interface by an RM or PE core. However, a DMA in a NoC interface contains only eight D-NoC UC resources. Both interfaces use a DMA engine to access memory and copy data. Regardless of whether a UC interface is used, it is necessary to allocate a D-NoC Tx resource and configure the Tx to send data. Additionally, it is necessary to allocate and configure D-NoC UC resources, if a UC interface is used.

Influences of Routing and Memory Type
Data transfer latencies between an IOS and a CC are influenced little by routing. This involves preparing two interfaces (Tx and UC), three routes (direct link, cross-link, and detour route), and two memory locations in which the transferred data are allocated. As shown in Figs. 3.2, 3.3, 3.4, and 3.5, end-to-end latency scales exhibit a linear relation with data size, and there are no significant differences between the three routes with respect to data transfer latency. This result is important in a torus topology NoC because the number of minimum steps exceeds that in a mesh topology. It is observed that queuing in NoC routers and hardware distance on a NoC are not dominant factors for latency. The router latency, the time taken in transmitting and receiving transactions in an RM,

exceeds those of other transactions. Additionally, it is briefly recognized that the speed of the UC exceeds that of the Tx. The data are arranged as shown in Figs. 3.6 and 3.7 to facilitate a precise analysis with respect to the interface and memory location. In those figures, only the cross-link from the IOS to CC5 is accepted because routes do not influence latency. To facilitate intuitive recognition, two kinds of figures are arranged: a logarithmic and a linear axis.

In the Tx interface, DDR causes a large increase in latency. The time taken by the DDR is twice that of the IO SMEM as shown in Fig. 3.6. This is due to the memory access speed characteristics of DRAM and SRAM. In the Tx interface, it is necessary for an IO core on an IOS to operate the DMA in the IOS NoC interface. This is attributed to the fact that the core is involved in processing. The speed of the data transfer latency between CCs exceeds that between an IOS and a CC. This result indicates that the MPPA-256 is optimized for communication between the CCs.

With respect to the UC interface, the latency is not significantly affected by the location at which the transferred buffer is allocated (i.e., DDR or SMEM). Similar latency characteristics are observed in Fig. 3.6. In the case of the UC interface, an IO core on the IOS does not involve a DMA transaction. A UC in the NoC interface executes a programmed thread sending data. This evaluation result suggests that the slow access speed of the DDR is not significant in the case of the UC. In a manner similar to that of the Tx interface, the speed of the data transfer latency between CCs exceeds that between an IOS and a CC.

3.0.2 Matrix Calculation

Situations and Assumptions

In the evaluation, the matrix calculation time and parallelization potential of MPPA-256 are clarified. Matrix calculations are conducted in an IOS and CCs. Three computing situations are considered as shown in Fig. 3.8. The first situation involves computing in the IOS where four cores are available. To analyze memory access characteristics, a matrix buffer is allocated in the IO DDR and SMEM. The second situation involves computing in a CC in which 16 cores are available. The third situation involves offload-computing using an IOS and four CCs. Parallelized processing is executed with four CCs and SMEMs. A few cores in the IOS and CC manage the parallelized transaction. The method can handle large amount of data in which one cluster is not sufficient, because buffer capacity is not limited to 2 MB in the SMEM. Parallelized processing and the total capacity of the SMEM are superior to single IOS or CC computations. With respect to the IOS, the application can handle large capacity data only in the DDR. However, in this method, distributed memories are used to deal with large capacity data in the SMEM. Thus, it is necessary for IOS and CC cores to access matrix buffers without cache to avoid cache coherency difficulties. To facilitate faster data transfer, a portion of the matrix buffer is transmitted in parallel as shown in Fig. 3.8.

Matrix calculation time is analyzed with parallelization and memory allocation. Additionally, the influences of a cache are analyzed because cache coherency is an important

issue in many-core systems. There are several cases in which applications must access specific memory space without a cache because MPPA does not guarantee cache coherency between PEs. With respect to the given assumptions, the maximum total buffer size is 1 MB, and thus three matrix buffers are prepared, each of size 314 KB. Matrix A and Matrix B are multiplied, and the result is stored in Matrix C. The total for the three matrices is set as approximately 1 MB. We assume that the remainder of the SMEM (1 MB) is occupied with system software and applications in the CC.

Influences of Cache and Memory Type

First, matrix calculation time with the cache in the IOS and CC is depicted in Fig. 3.9. There are almost no differences between the IO DDR, IO SMEM, and CC SMEM due to the cache. A 128 KB data cache in the IOS works well and compensates for the DDR delay. Additionally, it is observed that calculation time scales exhibit a linear relation with the number of threads. This corresponds to ideal behavior with respect to parallelization.

Second, matrix calculation time without a cache in the IOS and CC is shown in Fig. 3.10. The absence of a cache, results in a fourfold increase in the DDR and a large difference arises with respect to the SMEM. Another notable result is that calculation speed in the CC SMEM exceeds that of the IO SMEM. This characteristic is hidden in the calculation with the cache. The computing cores physically involve the same cores in the IOS and CC, and thus it is considered that the characteristics and physical arrangement of the SMEM exert a significant effect. This is an interesting result since there is a large difference that cannot be ignored. It is also observed that calculation time exhibits a linear relation with the number of threads. Furthermore, in the CC SMEM, the calculation speed without the cache exceeds that with cache. This result is contrary to intuition, and a cache line problem is conceivable. When a small data cache (8 KB) in a PE of the CC does not function adequately and an application always misses the cache, memory access will pay the time for a noncached data access and the cost to refill the cache line. As a result, the memory access speed without the cache exceeds that with it.

Four CCs' Parallelization

Finally, matrix calculation with offload-computing in an IOS and CCs is shown in Figs. 3.11 and 3.12. In this case, it is assumed with respect to the calculation of large matrices that the total capacity exceeds 1 MB. The offloading result is compared with the IO DDR (cached) owing to the aforementioned assumption. The aggregate calculation is obtained by offloading on the four CCs to perform a multiplication of a tile of Mat A and a tile of the transpose of Mat B. This produces an overhead irrespective of the number of threads as shown in Figs. 3.11 and 3.12.

However, the speed involved in offloading the result exceeds that of the IO DDR (cached). The result indicates several important facts. First, D-NoC data transfer produces little overhead latency. Second, the speed of DMA memory access to DDR exceeds

that of the IO core's memory access, even if target memory is allocated on the DDR. In the offloading case, a DMA accesses matrix buffers on the DDR and transfers the buffers from the IO DDR to each CC SMEM. Subsequently, PEs in the CC access matrix buffer the calculation without a cache. The overhead of data transfer and DMA memory access is small, and thus parallel data transmission and distributed memory are practical in the case of MPPA-256. The impact of offloading increases when the matrix is large as shown in Fig. 3.12. Only a portion of the matrix is allocated in CCs, and thus it is possible to handle larger matrix buffers.

Additionally, 640 KB matrices are prepared, and matrix calculation are evaluated with offload-computing. The speed of the offloading result exceeds that of the IO DDR result with respect to the 314 KB matrices in Fig. 3.11. In these offloading evaluations, each CC concurrently transmits calculation results to the IOS. When Matrix C in which calculation results are stored is allocated in the DDR, the NoC router's FIFOs sometimes overflow and cause an error due to memory access delay of the DDR. The transmission protocol would ideally be expected to prevent this error, and flow control is intended as future work for MPPA-256. Currently, to avoid this error, Matrix C must be allocated in the IO SMEM. Note that the above evaluation results when Matrix C is allocated in the DDR.

3.0.3 Practical Application

This work adopts a portion of a self-driving system and this section demonstrates the parallelization potential of the MPPA-256. We selected an algorithm for vehicle self-localization written in C++ in Autoware, open-source software for urban self-driving [22], and a parallelized part of it. The self-localization adopts the normal-distribution transform matching algorithm [23] implemented in the Point Cloud Library [24]. A diagram depicting self-localization is shown in Fig. 3.13.

The self-localization algorithm is composed primarily of the *computeTransform* function which searches for several nearest neighbor points for each scan query and calculates a matching transformation. This evaluation parallelized a part of *computeTransform* onto 16 CCs and the remainder of the algorithm was executed in parallel on the IOS with its

four cores. To parallelize the remainder of *computeTransform* in CCs, the algorithm of the nearest neighbor search must be redesigned because the data to be searched exceeds 1 MB. Redesigning this algorithm is reserved for future work, and there is room for improvement through the parallelization potential of the MPPA-256.

As shown in Fig. 3.14, the evaluation of the parallelized self-localization algorithm indicates the average execution time for each convergence and demonstrates that the parallelization accelerates the *computeTransform* process. The query can be assumed to be 10 Hz in many automated driving systems. Thus, this tuning successfully meets the deadline. This parallelized algorithm was executed on simulated and real car experiments in our test course and worked successfully. The steering, accelerator, and brake are automatically controlled based on the results of MPPA-256. A demonstration video of the adaptation of the parallelized self-localization algorithm to Autoware on eMCOS can be seen at: https://youtu.be/wZyqF90c5b8

3.0.4 Lessons Learned

Thus far, in Section 3, we have quantitatively clarified the characteristic of data transfer and parallel computing on NoC-based embedded many-core platforms. We can obtaine insight and guidelines for users and developers of NoC-based embedded many-core platforms through MPPA-256.

From evaluations of D-NoC data transfer, we can learn two lessons: the influences of NoC routing and DMA. First, data transfer latencies between clusters are hardly influenced by routing for users as shown in Figs. 3.2, 3.3, 3.4, and 3.5. Software transactions of transmitting and receiving in routers and RM are dominant factors for latencies. This is understandable owing to the minimized traffic conflicts of the evaluation setting. However, when the influence of routing comes to intensive concurrent traffic over a large portion of the network, different routes might still have nontrivial impacts on the end-to-end data transfer delay, especially for a detour route of a longer length. Second, in the IOS, the latency of a UC is not significantly affected by the memory location, the DDR or SMEM,

at which the transferred buffer is allocated from the evaluation of Fig. 3.6. The merit of the UC is profitable for users because this is a common situation for transferrinng data from the DDR to the CC SMEM using the UC, especially for large amount of data.

From microbenchmarks of matrix calculation, we can learn three lessons of memory access: characteristics of the SMEM, influences of a cache, and data flow control on D-NoC. First, the memory access speed of the CC SMEM exceeds that of the IO SMEM as shown in Fig. 3.10. There is a significant difference, which is a major reason to work actively in the CC. Second, we indicate quantitatively that the calculation speed without the cache in the CC SMEM in Fig. 3.10 exceeds that with the cache in Fig. 3.9. It is generally known that there is cache overhead when a miss hit occurs frequently, but it is notable that there is a substantial difference that cannot be ignored by the influences of a cache line. Third, when data are transferred in parallel from multiple CCs to the IO DDR, NoC routers' FIFO sometimes overflows owing to memory access delays of the DDR. This would ideally be expected to prevented by the transmission protocol and flow control.

From the practical application viewpoint, since we parallelize the vehicle self-localization algorithm of the self-driving system, NoC-based embedded many-core systems can be used practically in real environments. We applied parallel data transfer from an IOS to CCs and parallel computing on the IOS and CCs by using the CC SMEM as scratch pad memory. We also conducted demonstration experiments with real environments and confirmed the practicality of NoC-based embedded many-core systems.

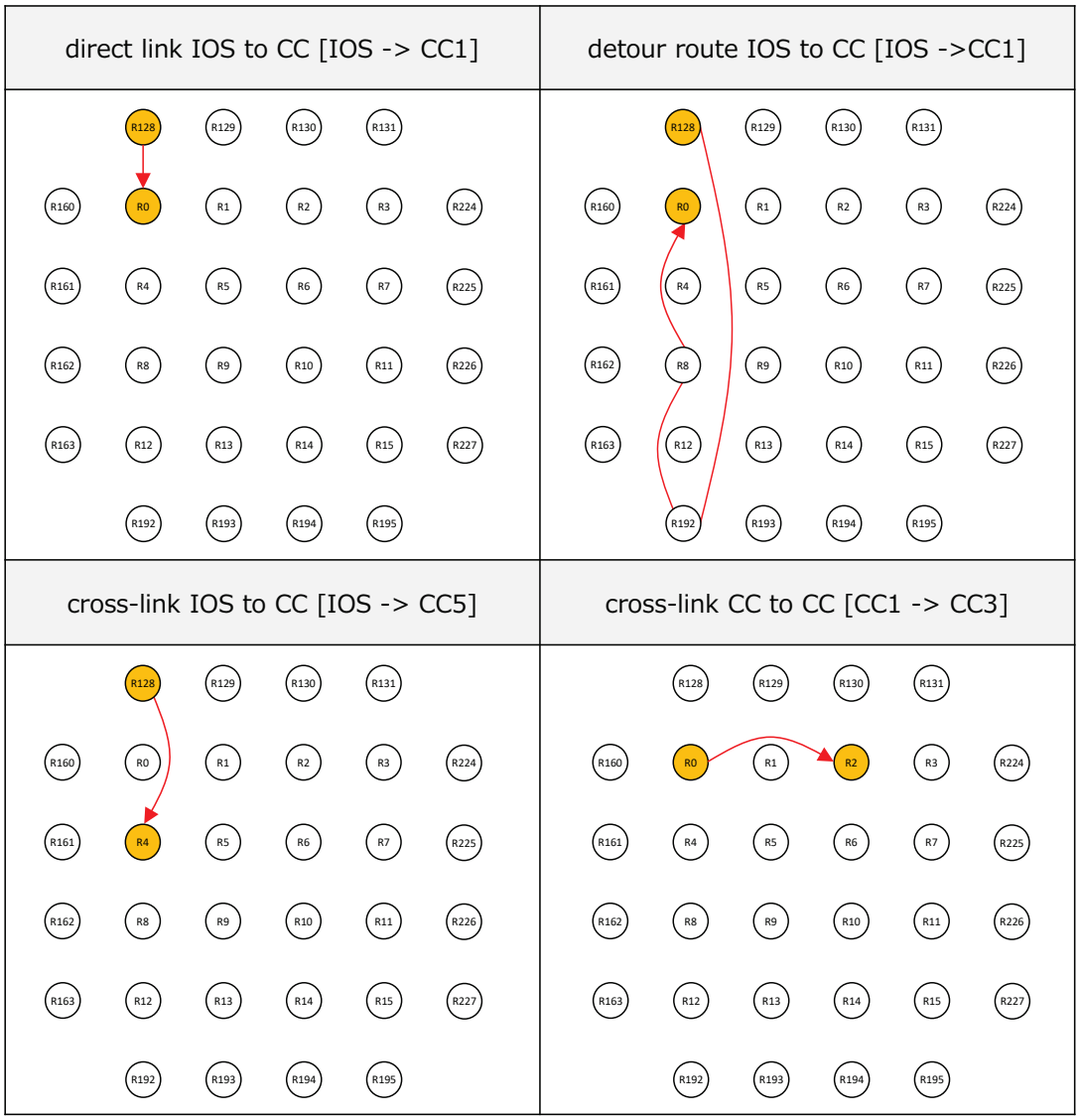

Fig. 3.1 Four D-NoC routes used in the evaluation.

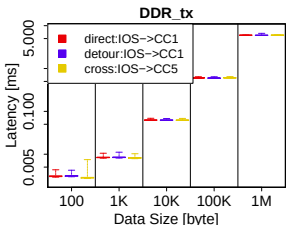

Fig. 3.2 Data transfer with Tx from IO DDR to CC.

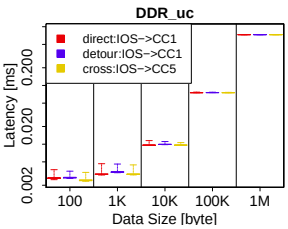

Fig. 3.3 Data transfer with UC from IO DDR to CC.

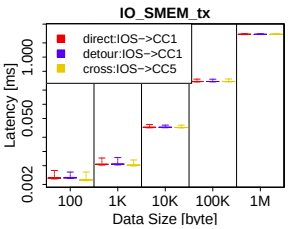

Fig. 3.4 Data transfer with Tx from IO SMEM to CC.

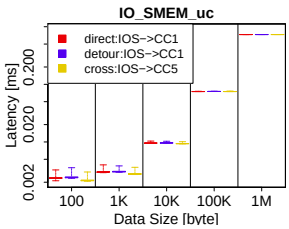

Fig. 3.5 Data transfer with UC from IO SMEM to CC.

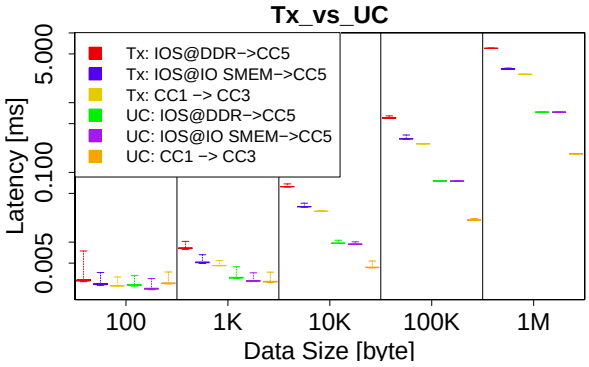

Fig. 3.6 Data transfer with Tx/UC (logarithmic axis).

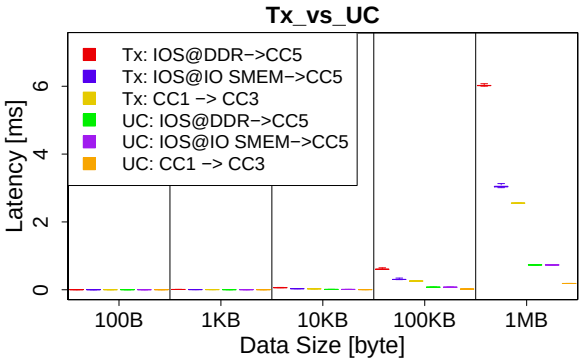

Fig. 3.7 Data transfer with Tx/UC (linear axis).

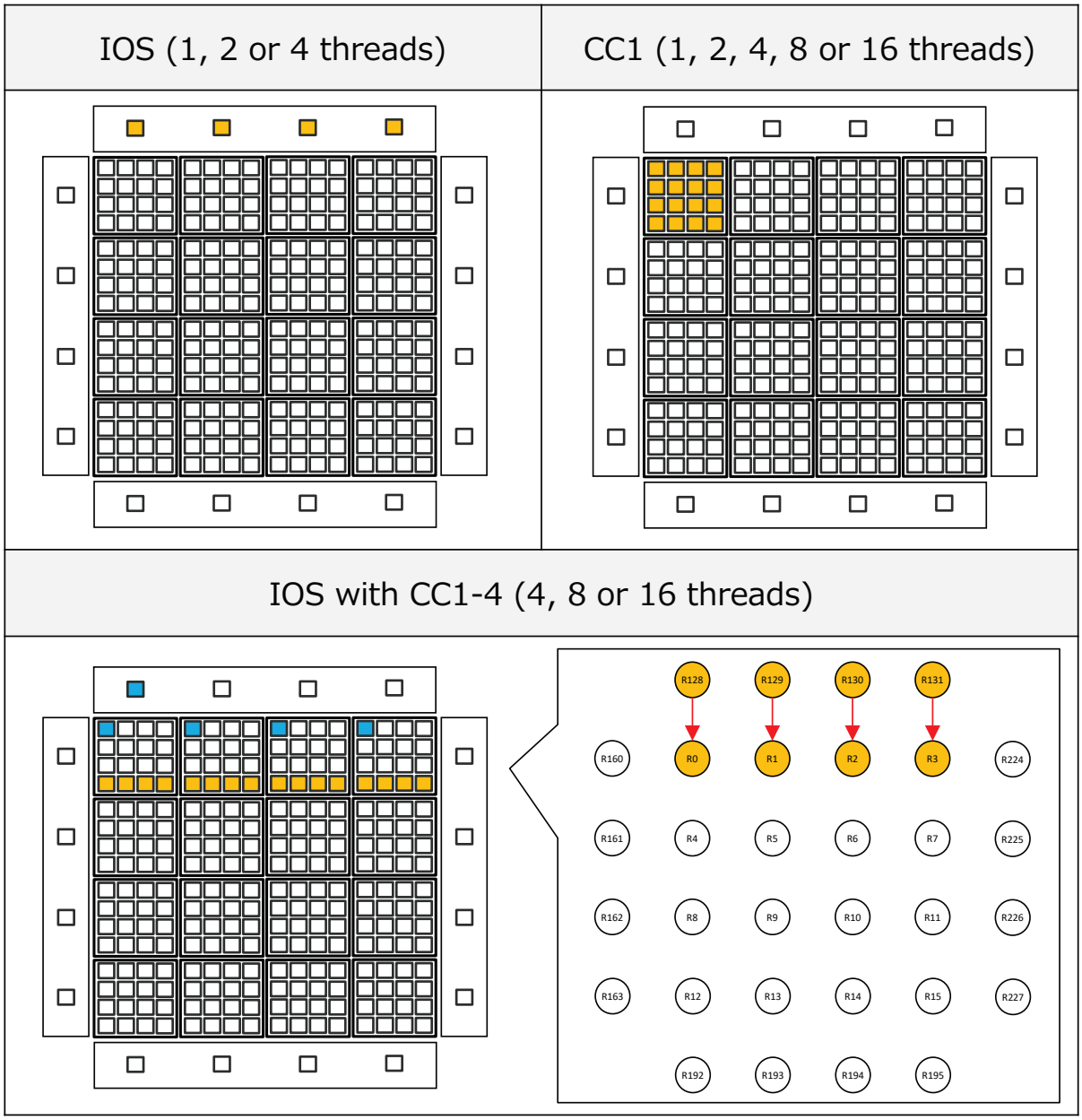

Fig. 3.8 Matrix calculation situations.

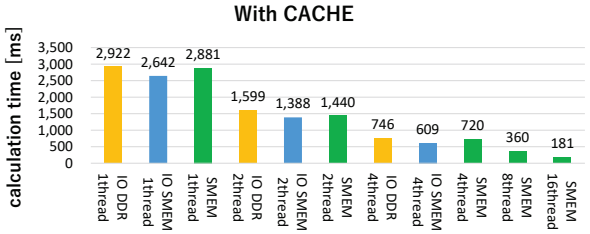

Fig. 3.9 Matrix calculations in IOS and CC with cache.

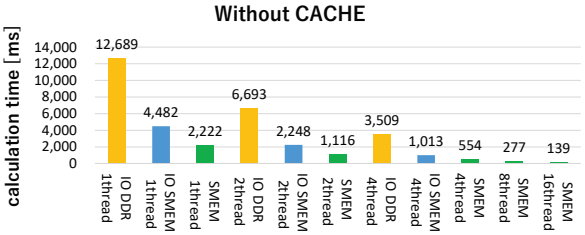

Fig. 3.10 Matrix calculations in IOS and CC without cache.

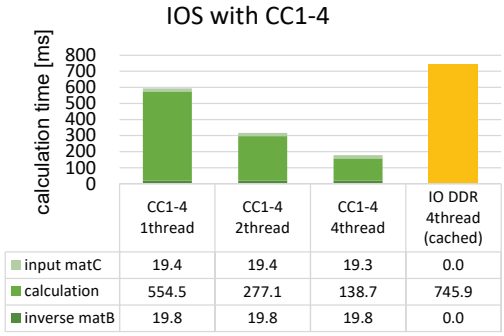

Fig. 3.11 Matrix calculations with offload computing (314 KB matrix x 3).

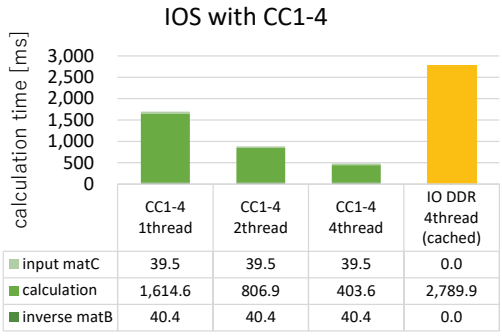

Fig. 3.12 Matrix calculations with offload computing (640 KB matrix x 3).

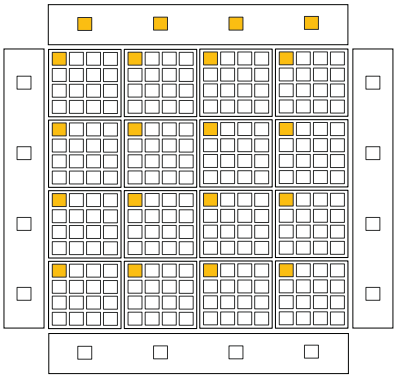

Fig. 3.13 A situation of vehicle self-localization execution.

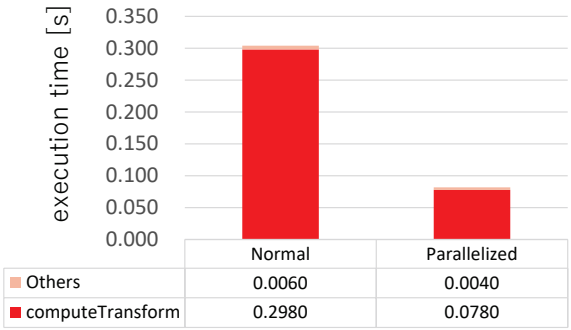

Fig. 3.14 Vehicle self-localization of the partially parallelized self-driving application.

Chapter 4

Related Work

This section compares many-core platforms and discusses previous work related to multi-/many-core platforms. First, comparison of many-core platforms to other platforms is discussed. Second, the Kalray MPPA-256 which this work focuses on is compared to other COTS multi-/many-core components, and we summarize the features of MPPA-256. Finally, discussions of previous work and comparisons with them are described.

Table 4.1 summarizes the features of many-core platforms with those of other platforms. For instance, the GPU is a powerful device to enhance computing performance and has great potential in specific areas (for e.g., image processing, and learning). However, th GPU is mainly used for a specific purpose and its predictability is not suitable for real-time systems. It is difficult to use a GPU for many kinds of applications and to guarantee its reliability due to the GPU architecture. Many-core processors based on CPU are significantly superior to GPU with respect to software programmability and timing predictability. Additionally, it is commonly known that many-core platforms such as MPPA-256 involve a reasonable power consumption [14]. In contrast, the GPU consumes a significant amount of power and generates considerable heat. This is a critical problem for embedded systems. FPGAs are also high-performance devices when compared to CPUs. They are efficient in terms of power consumption. FPGAs guarantee predictability and efficient processing. However, FPGAs are difficult for software developers and are not a substitute for CPU since their software model is significantly different from that of CPU. Many-core platforms can potentially replace single/multi core CPU as they possess ease of programming and scalability.

Based on the aforementioned background, many COTS multi-/many-core components are developed and released by several vendors. (e.g., MPPA-256 by Kalray, [7], Tile-Gx by Tilera [8], [9], Tile64 by Tilera [10], and Xeon Phi by Intel [11], [12], Single-chip Cloud Computer (SCC) by Intel [13]). The present work focuses on the Kalray MPPA-256, which is designed for embedded applications. Kalray [7] presented clustered many-core architectures on the NoC that pack 256 general-purpose cores with high energy efficiency.

MPPA-256 is superior to other COTS multi-/many-core components in terms of the scalability of the number of cores and the power efficiency, as shown in Table 4.2. In terms of scalability, MPPA-256 uses 256 cores, whereas other COTS multi-/many-core components have 64 cores or the number of cores around it. This scalability of cores is

attributed to the NUMA memory architecture; each cluster of 16 cores contains its own local shared memory. The precise hardware model is described in Section 2.0.1. When all cores share the global DDR memory as in other platforms excluding MPPA-256, specific bus/network routes receive extremely large loads and memory access contention frequently occurs. Local shared memory reduces the above problems and helps the scalability of the number of cores. However, the NUMA memory architecture restricts the capacity of the memory and requires a data copy from the DDR with NoC. This restriction makes the use of existing applications difficult especially in the case of applications that require more memory. As a result, owing to the NUMA memory architecture, the portability of code porting to MPPA-256 is inferior to that of other COTS platforms, as shown in Table 4.2.

In terms of power efficiency, MPPA-256 realizes superior energy efficiency despite its large number of cores [14]. The total clock frequency per watt is the highest of the current COTS multi-/many-core components. The power consumption of the MPPA processor ranges between 16 W at 600 MHz and 24 W at 800 MHz. We must distinguish the COTS multi-/many-core components according to their requirements with reference to Table 4.2. MPPA-256 is typically accepted with respect to many-core platforms and the model has been used in previous work [1], [25], [2], [16].

Previous work has examined real-time applications on many-core platforms, including MPPA-256. Multiple opportunities and challenges of multi-/many-core platforms are discussed in [3]. The shift to multi-/many-core platforms in real-time and embedded systems is also described.

Based on the above background, several task mapping algorithms for multi/many-core systems have been proposed [25], [26], [2]. Airbus [2] proposes a method of directed acyclic graph (DAG) scheduling for hard real-time applications using MPPA-256. In [26], a mapping framework is proposed on the basis of AUTOSAR which is applied as a

standard architecture to develop automotive embedded software systems [27]. AUTOSAR task scheduling considering contention in shared resources is presented in [1].

By examining the above mapping algorithms of real-time applications, previous work [28], [15], [14], [16] has analyzed the potential of MPPA-256 and data transfer with NoC, as shown in Table 4.3. MPPA-256 is introduced and its performance and energy consumption are reported in [14]. However, this report contains few evaluations and does not refer to data transfer with NoC and memory access characteristics. Data transfer with NoC in MPPA-256 is described, and NoC guaranteed services are analyzed in [28] and [15]. While the theoretical analysis is thorough in these works, the practical evaluations are poor and parallel data transfer is not referred to. The authors of Ref. [16] focused on the predictable composition of memory access. An analysis of their work identified the external DDR and the NoC as the principal bottlenecks for both the average performance and the predictability on platforms such as MPPA-256. Although the analysis examined the memory access characteristics of the external DDR and provided notable lessons, a solution for the DDR bottleneck was not examined and practical evaluations were lacking.

Table 4.1 Comparison of Many-core Platform to CPU, GPU, and FPGA

	performance	power/heat	real-time	software	costs development	multiple
CPU		L	✓	✓	✓	L
GPU	✓			L	✓	
FPGA	✓	L	L		L	
Many-core Platform	✓	✓	✓	✓	L	✓

∗In a table, “L” means “limited”.

Table 4.2 Comparison of Many-core Platforms

	scalability	power efficiency	code transplant
Kalray MPPA-256 [7]	✓	✓	L
Tilera Tile series [10]		L	✓
Intel Xeon Phi [11] [12]			✓
Intel SCC [13]			✓

Table 4.3 Comparison of Previous Work

	performance analysis	data transfer analysis with NoC	memory access characteristics	real applications
Kalray clusters calculate quickly [14]	L			
Network-on-Chip Service Guarantees [15]		✓		
Predictable composition of memory accesses [16]		✓	✓	
this paper	✓	✓	✓	✓

Chapter 5

Conclusions

In this work, we conducted quantitative evaluations of data transfer methods on NoC technology, microbenchmarks with matrix calculation, and a practical application with NUMA many-core platforms such as MPPA-256. Evaluations indicate latency characteristics on NoC platforms, influences of data allocation, and the scalability of parallelization. Our experimental results will allow system designers to select appropriate system designs. Last, parallelization of a real application proved the practicality of NoC-based embedded NUMA many-core platforms.

In future work, we will use benchmark applications such as that in [29] for further analysis. In addition, considering above the results, we will port real-time systems such as that in [30] on CCs, and we propose the parallelization of memory intensive algorithms, such as the nearest neighbor search in Section 3.0.3.

Acknowledgment

This paper was partly supported by Toyota Motor Company, eSOL, and Kalray. In addition, this research was partially supported by JST, PRESTO.

References

[1] M. Becker, D. Dasari, B. Nicolić, B. Akesson, V. Nélis, and T. Nolte, "Contention-free execution of automotive applications on a clustered many-core platform," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 14–24.

[2] Q. Perret, P. Maurère, É. Noulard, C. Pagetti, P. Sainrat, and B. Triquet, "Mapping hard real-time applications on many-core processors," in *Proc. of International Conference on Real-Time Networks and Systems (RTNS)*, 2016, pp. 235–244.

[3] S. Saidi, R. Ernst, S. Uhrig, H. Theiling, and B. D. de Dinechin, "The shift to multicores in real-time and safety-critical systems," in *Proc. of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015, pp. 220–229.

[4] Q. Perret, P. Maurère, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet, "Temporal isolation of hard real-time applications on many-core processors," in *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–11.

[5] M. Becker, K. Sandström, M. Behnam, and T. Nolte, "Mapping real-time tasks onto many-core systems considering message flows," in *Proc. of the Work-in-Progress Session of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.

[6] B. Paolo, B. Marko, N. Capodieci, C. Roberto, S. Michal, H. Pemysl, M. Andrea, G. Paolo, S. Claudio, and M. Bruno, "A software stack for next-generation automotive systems on many-core heterogeneous platforms," *Microprocessors and Microsystems*, vol. 52, no. Supplement C, pp. 299 – 311, 2017.

[7] B. D. de Dinechin, D. Van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–6.

[8] C. Ramey, "TILE-Gx100 manycore processor: Acceleration interfaces and architecture," in *Proc. of IEEE Hot Chips Symposium (HCS)*. IEEE, 2011, pp. 1–21.

[9] R. Schooler, "Tile processors: Many-core for embedded and cloud computing," in *Proc. of Workshop on High Performance Embedded Computing*, 2010.

[10] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown *et al.*, "Tile64-processor: A 64-core soc with mesh interconnect," in *Proc. of IEEE Solid-State Circuits Conference (ISSCC)*. IEEE, 2008, pp. 88–598.

[11] G. Chrysos, "Intel® Xeon Phi Coprocessor-the Architecture," *Intel Whitepaper*, 2014.

[12] G. Chrysos and S. P. Engineer, "Intel Xeon Phi coprocessor (codename knights corner)," in *Proc. of Hot Chips Symposium (HCS)*, 2012.
[13] M. Baron, "The single-chip cloud computer," *Microprocessor Report*, vol. 24, p. 4, 2010.
[14] D. Kanter and L. Gwennap, "Kalray clusters calculate quickly," *Microprocessor Report*, 2015.
[15] B. D. de Dinechin and A. Graillat, "Network-on-Chip Service Guarantees on the Kalray MPPA-256 Bostan Processor," in *Proc. of International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems (AISTECS)*, 2017.
[16] Q. Perret, P. Maurère, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet, "Predictable composition of memory accesses on many-core processors," in *Proc. of European Congress on Embedded Real Time Software and Systems (ECRTS)*, 2016.
[17] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proc. of IEEE Design Automation Conference (DAC)*, 2001, pp. 684–689.
[18] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob *et al.*, "An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS," in *Proc. of IEEE International Solid-State Circuits Conference (ISSCC)*, 2007, pp. 98–99.
[19] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee *et al.*, "The raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.
[20] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): The case for a scalable operating system for multicores," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 76–85, 2009.
[21] A. Schüpbach, S. Peter, A. Baumann, T. Roscoe, P. Barham, T. Harris, and R. Isaacs, "Embracing diversity in the barrelfish manycore operating system," in *Proc. of the Workshop on Managed Many-Core Systems*, 2008, p. 27.
[22] "Autoware: Open-source software for urban autonomous driving," https://github.com/CPFL/Autoware.
[23] M. Magnusson, "The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection," Ph.D. dissertation, Örebro universitet, 2009.
[24] "Point Cloud Library (PCL)," http://pointclouds.org/.
[25] T. Carle, M. Djemal, D. Potop-Butucaru, R. De Simone, and Z. Zhang, "Static mapping of real-time applications onto massively parallel processor arrays," in *Proc. of IEEE International Conference on Application of Concurrency to System Design*, 2014, pp. 112–121.
[26] H. R. Faragardi, B. Lisper, K. Sandström, and T. Nolte, "A communication-aware solution framework for mapping AUTOSAR runnables on multi-core systems," in *Proc. of IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–9.

[27] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange, "Autosar–a worldwide standard is on the road," in *Proc. of International VDI Congress Electronic Systems for Vehicles (ELIV)*, vol. 62, 2009.

[28] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, "Guaranteed Services of the NoC of a Manycore Processor," in *Proc. of ACM International Workshop on Network on Chip Architectures (NoCArc)*, 2014, pp. 11–16.

[29] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads," in *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2010, pp. 1–11.

[30] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the Performance of ROS2," in *Proc. of ACM International Conference on Embedded Software (EMSOFT)*, 2016, pp. 5:1–5:10.

Publication List

International Conference Proceedings

[1] Y.Maruyama, S.Kato, and T.Azumi, "Impact of DDS on ROS2," *In Proceedings of the 19th IEEE International Symposium on Real-Time Computing (ISORC2016)*, 2016. (submitted)

[2] Y.Maruyama, S.Kato, and T.Azumi, "Preliminary Evaluation of ROS2," *WiP session of the ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS2016)*, 2016. (accepted)