Exploring Data Transfer with Network-on-Chip based Many-Core Processors

Yuya Maruyama
Graduate School of Engineering Science
Osaka University

Takuya Azumi
Graduate School of Engineering Science
Osaka University

ABSTRACT

The evolution of a next generation computing platform oriented toward multi/many cores is unavoidable because the demand of high-performance computation is increasing with reasonable power consumption even in embedded real-time systems. The Multi-Purpose Processing Array (MPPA) 256 developed by Kalray is one of many-core processors platforms and packs 256 general-purpose cores with high energy efficiency. The clusters of cores share a local memory, and a DMA-capable network-on-chip (NoC) connects the clusters. In this paper, we investigate and characterize currently-achievable data transfer methods of cutting-edge many-core technology based on NoC. Previous work has not quantitatively examined the transport latencies and not clarified the potential and constraints for low-latency many-core computing. In order to address this problem, we illustrate the evaluations of end-to-end latencies and the calculation potential of many-core computing with parallel data transfer. Our experimental results allow system designers to choose appropriate data transfer methods based on the requirement of their latency-sensitive many-core applications beyond an intuitive expectation. By highlighting the many-core computing capabilities, we explore the performance characteristics of currently-achievable data transfer methods for many-core computing based on NoC.

Keywords

Many-core; Multicore; Network-on-Chip

1. INTRODUCTION

The evolution of a next generation computing platform oriented toward multi/many cores is unavoidable to satisfy the demand for increasing computation in conjunction with reasonable power consumption. Recently, the computational ability of single core processors has reached its limit and thus the applicability of Moore's law [23] is unclear. Multi/many core architecture is an important trend in recent years as it integrates cores to realize high-performance and general-purpose computing with low power consumption. Hence, high energy efficiency is a superior feature of multi/many core platforms.

As detailed in the preceding paragraph, the demand for increasing computation in conjunction with reasonable power consumption drives the need for multi/many core architecture in several domains. Real-time embedded systems are an example in which multi/many core platforms are adapted since they face increasing processing requirements. Extant studies have examined numerous applications of multi/many cores platforms [4], [30], [26], [25], [5].

For example, automotive systems involve various applications and are sometimes characterized by demands for high-performance computing. Automotive applications are responsible for several control systems such as the powertrain, chassis, steering wheel, driver assistance and user interface. Advanced driver assistance systems are characterized by increasing complexity and computational requirements and necessitate intelligence such as an autonomous driving system. Moreover, they also require significant energy efficiency and cost reduction. Although modern automotive systems are composed of several Electronic Control Unit (ECU) managing subsystems, there is a shift from numerous scattered ECUs to hierarchical multi/many core Domain Controllers (DC). Evidently, a hybrid scenario is also applicable. Specifically, DCs combine performance with low power consumption to realize high parallel processing and efficient data transport. Furthermore, flexible scalability of a many-core platform acilitates development efficiency.

A related example involves avionics systems that also include various applications and require cost efficiency. Multi/many core platforms can be applied in the avionics domain due to their highly integrated characteristics and increasing processing demands. Low power consumption of cores leads to lower cooling requirements, and this is a critical issue for avionics. Multi/many core platforms reduce energy consumption as well as the costs and weight. Weight reduction is the most important factor for avionics, and integration with a multi/many core platform reduces the amount of processing boards and cooling units. Thus, the utilization of an integrated platform results in space, weight and cost reductions.

Hence, multi/many core platforms are fabricated and released as commercial off-the-shelf (COTS) multicore components. Increasing research attention has focused on multi/many core platforms. The Multi-Purpose Processing Array (MPPA) 256 developed by Kalray [14], [13], [17], Single-chip Cloud Computer (SCC) developed by Intel [3], and Tile64 devel-

EMSOFT'16, October 01-07, 2016, Pittsburgh, PA, USA
© 2016 ACM. ISBN 978-1-4503-4485-2/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2968478.2968502

oped by Tilera [6] include the clustering of many-core architectures in which cores are mapped closely. The clusters of cores are capable of performing separate independent applications with respect to the desired power envelope of embedded applications. Kalray's MPPA-256 packs 256 general-purpose cores. This significantly exceeds the number of cores in other COTS, and it targets embedded systems, high-performance computing (HPC), image processing, and networking due to the high-performance per watt. Intel's Xeon Phi [9], [10] is an HPC accelerator. Xeon Phi and SCC are based on x86 architecture and its targeted use corresponds to data centers and workstations.

Despite the emergence of the need for multi/many core platforms, several difficulties persist in the adaptation of these platforms to real-time embedded systems [4], [30]. These difficulties are caused by the hardware architecture and strict requirements of embedded systems. A difficulty involves the sharing of numerous resources (e.g. memory subsystems and I/O devices) by cores. Parallelized complex processes share some resources, and this leads to the frequent occurrence of conflicts. Cache coherency is also a critical problem owing to their numerous cores. These difficulties disturb predictable timing behavior and software analysis. Thus, the timing requirement of real-time embedded systems continues to warrant solutions. The impact of integrating real-time applications in multi/many core platforms, is not completely understood to date. This is a critical issue because embedded systems have requirements for reliable and predictable behavior.

Contributions: This study focused on clarifying performance characteristics of currently-achievable data transfer methods for many-core computing based on Network-on-Chips (NoC), such as MPPA-256 while examining parallel data transfer methods. We reveal the advantage and disadvantage of many-core computing based on NoC in a quantitative way leading the following contributions:

- Evaluations of data transfer on DMA-capable NoC quantitatively characterize end-to-end latencies, which depend on routing and DMA configurations.

- The scalability of parallelization in many-core processors based NoC is observed in evaluations of matrix calculation in several situations and a real complex application such as a part of autonomous driving systems.

- Evaluations of parallel processing clarify the characteristics of memory access speed which is varied by where data is allocated and what accesses the memory.

To the best of the authors' knowledge, this is the first study that examines data transfer matters for many-core computing beyond an intuitive expectation to allow system designers to choose appropriate data transfer methods based on the requirement of their latency-sensitive many-core applications. It is expected that the findings of the study are potentially applicable for several types of many-core architectures as opposed to solely for MPPA-256. The contributions of this study can be used in low-latency many-core computing.

Organization: The remainder of this study is organized as follows. First, the system model considered in this study is discussed in Section 2 in which the hardware model, namely Kalray MPPA-256 Bostan, and the system model are presented. Second, Section 4 illustrates experimental evaluations. Subsequently, Section 5 examines related studies that focus on multi/many core systems. Finally, Section 6 presents the conclusions and directions for future research.

2. SYSTEM MODEL

This section presents the system model used throughout the study. The many-core model of Kalray MPPA-256 Bostan is considered. First, a hardware model is introduced in Section 2.1, and this is followed by a software model in Section 2.2.

2.1 Hardware Model

The MPPA-256 processor is based on an array of compute clusters (CCs) and I/O subsystems (IOSs) that are connected to nodes of Network-on-Chip (NoC) with a toroidal 2D topology (as shown in Figures 1, 2, and 3). The MPPA MANYCORE chip integrates 16 compute clusters and 4 IOS on NoC. The architecture of Kalray MPPA-256 is presented in this section.

2.1.1 I/O Subsystems (IOS)

MPPA-256 has the following four I/O subsystems (IOSes): North, South, East, and West IOSes. The North and South IOSes are connected to a DDR interface and an eight-lane PCIe controller. The East and West IOSes are connected to a quad 10Gb/s Ethernet controller. Two pairs of IOSes organize two I/O clusters (IOCs) as shown in Figure 1.

Each IOS comprises of quad core resource managers (RMs) and a network interface.

- **Resource Managers**: RM cores are connected to a 16-banked parallel shared memory with a total capacity (IO SMEM) of 2MB, as shown in the left side of Figure 1, The four RMs 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 RMs. Additionally, RM cores operate controllers for the PCIe, Ethernet, Interlaken, and other I/O devices. They operate the local peripherals, including the network interfaces with DMA. It is also possible to conduct an application run on the RMs.

- **Network Interface**: The network interface contains four DMA engines (DMA0-3) and four NoC routers as shown in the left side of Figure 1, and the IOS DMA engine manages transfers between the IO SMEM, the IOS DDR, and the IOS peripherals (e.g., PCIe interface and Ethernet controllers). The DMA engine transfers data between routers on NoC through NoC routers. The DMA engine has the following three NoC interfaces: a receive (Rx) interface, a transmit (Tx) interface, and a micro core (UC). A micro core is a network processor that can be programed to set threads sending data with a Tx interface. A UC can extract data from memory by using a programed pattern and send the data on the NoC. After it is initiated, this continues in an autonomous fashion without using a Processing Elements (PE) and an RM.

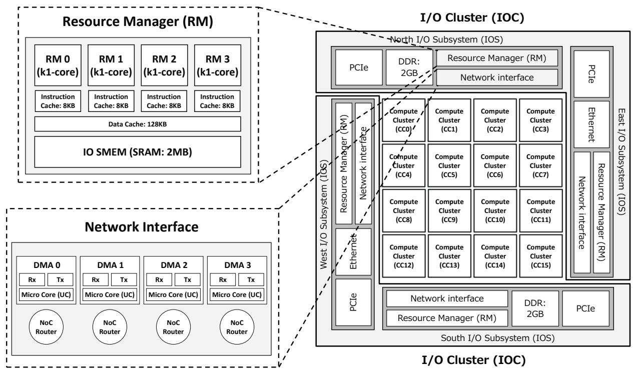


Figure 1: An overview of the architecture of the Kalray MPPA-256 Bostan.

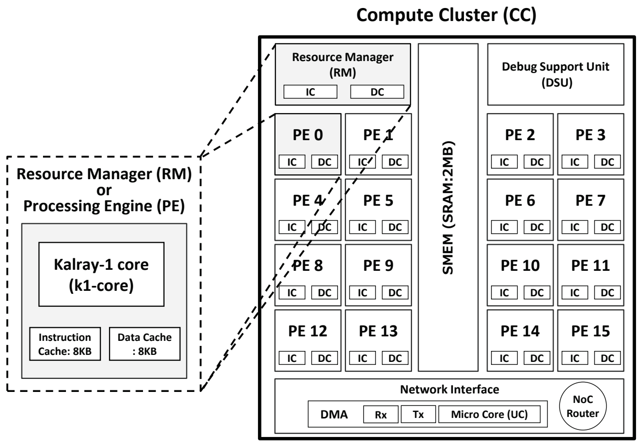


Figure 2: Compute Cluster architecture.

2.1.2 *Compute Clusters (CC)*

In MPPA-256, the 16 inner nodes of the NoC correspond to the CCs. Figure 2 illustrates the architecture of each CC.

- **Processing Elements and an RM**: In a CC, 16 processing elements (PEs) and a RM share 2 MB cluster local memory (SMEM) that is composed with 16 independent memory banks with 384 × 64bit. The capacity of each SMEM bank corresponds to 128 KB. The PEs are mainly used by users for parallel processing. Developers spawn computing threads on PEs. The PEs and an RM in CC correspond to the Kalray-1 cores, which implement a 32-bit 5-issue Very Long Instruction Word architecture with 16 W-600 MHz (typical) or 24 W-800 MHz. Each core is fitted with its own instruction and data caches. Each cache is a 2-way associative with a capacity of 8 KB. Thus, 17 k1-cores (a PE or the RM) share a multi-banked 2 MB SMEM.
- **A Debug Support Unit (DSU) and a Network Interface**: In addition to PEs and an RM, bus masters on SMEM correspond to a DSU and a DMA engine in a network interface. A DMA engine and a NoC router are laid out in a network interface. In a manner similar to IOS, the CC DMA engine also has the following three interfaces: an Rx, a Tx, and a UC. It is instantiated in every cluster and connected to the SMEM.

2.1.3 *Network-on-Chip (NoC)*

The 16 CCs and the 4 IOSs are connected by a NoC as shown in Figure 3. Furthermore, a NoC is constructed on the bus network and has routers on each node.

- **Bus Network**: Bus network connects nodes (CCs and IOSs) with torus topology [11] which has dual bands and involves a low average number of hops when compared to mesh topology [33], [32]. The network is actually composed of the following two parallel NoCs with bi-directional links (denoted by red lines in Figure 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. Functionally, NoC corresponds to a packet switched network. Data are packaged in variable length packets that circulate between routers in a wormhole manner in which packets are broken into small pieces called flits (flow control digits). The NoC traffic is segmented into packets, and each packet includes 1 to 4 header flits and 0 to 62 payload data flits.
- **NoC routers**: A node per compute cluster and four nodes per I/O subsystem holds the following two routers of its own: a D-NoC router and a C-NoC router Each RM on NoC node (CC or IOS) is associated with the fore-mentioned two NoC routers. Furthermore, DMA engines in a network 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 synchronizations. The NoC routers shown in Figures 1 and 2 illustrate nodes as R0-15, R128-131, R160-163, R224-227, and R192-195 in Figure 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 (CC or 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 rate of 600 MHz or 800 MHz, and therefore 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).

2.2 Software Model

Figure 4 shows the software stack used for Kalray MPPA-245 in the present study. The Kalray system is an extensible and scalable array of computing cores and memory. With respect to this type of a system, it is possible to map several programing models or runtimes such Linux, real-time operating system, POSIX API, OpenCL, and OpenMP. Each

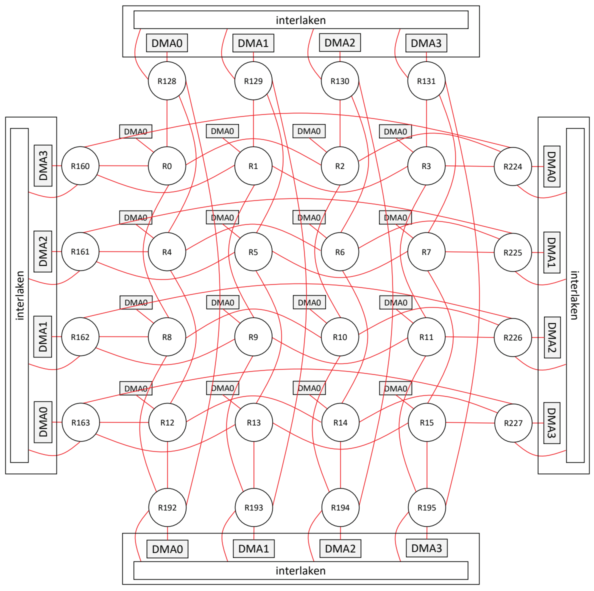


Figure 3: NoC connections (both D-NoC and C-NoC).

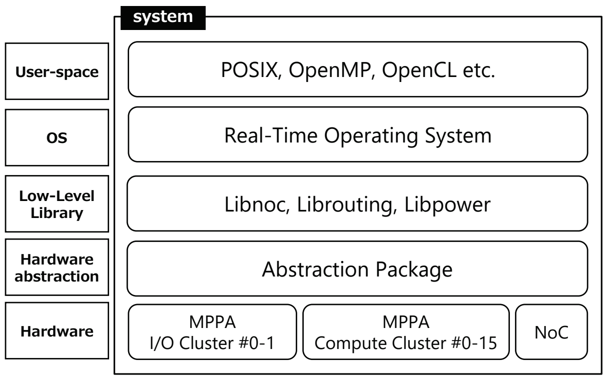

system
User-space
OS
Low-Level Library
Hardware abstraction
Hardware
POSIX, OpenMP, OpenCL etc.
Real-Time Operating System
Libnoc, Librouting, Libpower
Abstraction Package
MPPA I/O Cluster #0-1
MPPA Compute Cluster #0-15
NoC

Figure 4: The software stack of Kalray MPPA-256.

layer is described in detail.

In the hardware abstraction layer, an abstraction package abstracts hardware of a CC, IOS, and NoC. The abstraction package serves as a system that does not provide any services. The hardware abstraction is responsible for partitioning hardware resources and controlling access to the resources from the user-space operating system libraries. Additionally, the abstraction package retrieves resources allocated to a partition at any time. It sets-up and controls inter-partition communications as a virtual machine abstraction layer. The hardware abstraction runs on the dedicated RM core. All the services are commonly provided by a rich operating system (for e.g., virtual memory, interrupts, and schedule) that must be provided by user-space libraries. Consequently, each runtime or operating system implements its own services that are optimized to specific needs. This is because each programing model or runtime involves different requirements. A minimal kernel avoids wastage of resources and mismatched needs.

In a low-level library layer, the Kalray system also provides libraries for handling NoC. Additionally, NoC features such as routing and quality of service are set by the programmer. The Libnoc allows direct access to memory mapped registers for their configurations and uses. It is designed to cause a minimum amount of CPU overhead. It 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 including unicast (one target) modes or multicast (multiple targets) modes. As shown in Figure 3, routing on the torus network is statically conducted with its own policy. The Libpower enables spawning and waiting for the end of execution of a remote cluster.

Various operating systems support the abstraction package in the OS layer. The following Real-Time Operating System (RTOS) is introduced:

- **RTEMS**: RTEMS (Real-Time Executive for Multiprocessor Systems) is a full featured RTOS prepared for embedded platforms. It supports several APIs and standards, and most notably supports the POSIX API. The system provides a rich set of features, and an RTEMS application mostly corresponds to a regular C or C++ program that uses the POSIX API. Additionally, RTEMS can be built on the IOC.
- **NodeOS**: On 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 POSIX API to run on PEs on CC. First, NodeOS runtime starts on PE0 prior to before calling the user main function. Subsequently, pthread is called on other PEs.
- **eMCOS**: On both CC and IOS, eMCOS provides minimal programing interfaces and libraries. Specifically, eMCOS is a real-time embedded operating system developed by eSOL (a Japanese supplier for RTOS), and eMCOS is the world's first commercially available many-core RTOS for use in embedded systems. The OS implements a distributed micro-kernel architecture. This compact micro-kernel is equipped with only minimal functions. It enables applications to operate priority based message passing, local thread scheduling, and thread management on IOS as well as CC.

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

3. DATA TRANSFER FRAMEWORK

In this section, data transfer methods in MPPA-256 are explained. For scalability purposes, MPPA-256 accepts clustered architectures in which each cluster contains its own memory. 16 cores are packed as a cluster and they share 2 MB memory (SMEM) as shown in Figure 2. This avoids frequent memory contention due to numerous cores and helps in increasing the number of cores. However, the architecture constraints memory sthat can be directly accessed by the cores. In order to communicate with cores outside the cluster, it is necessary to transfer data between clusters through the D-NoC with network interfaces.

A Rx interface exists in the receiving side to receive data with DMA. It is necessary to allocate a D-NoC Rx resource and configure it to wait to receive the data. A DMA in a network interface contains 256 D-NoC Rx resources. Two interfaces, namely a Tx interface and a UC interface as explained in Sections 2.1.1 and 2.1.2, 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. It executes programed pattern and sends data through the D-NoC without a PE and an RM. The UC interface results in faster data transfer when compared to that in the Tx interface. However, a DMA in a network interface contains only 8 D-NoC UC resources. Both interfaces use a DMA engine to access memory and copy data. Irrespective of whether or not a UC interface is used, it is necessary to allocate a D-NoC Tx resource and configure it to send data. Additionally, it is necessary to allocate and configure D-NoC UC resources if a UC interface is used.

4. EVALUATIONS

This section involves examining two types of evaluations, namely 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 while dealing with large data.

4.1 D-NoC Data Transfer

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 Figure 5. The routes on D-NoC map (Figure 3) contains various connections between routers, namely a direct link, a cross link, and a flying link. With respect to case of routes from the IOS routers to the CC routers, transmitted data is allocated in DDR or IO SMEM. The CC only includes SMEM as shown in Figure 2. A low-level library is directly used to transfer data with D-NoC. 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 CC SMEM corresponds to 2 MB, and thus it is assumed that the appropriate communication buffer size corresponds to 1 MB. Given the assumption, the other memory area corresponds to the application, library, and operating system. End-to-end latencies are measured 1,000 times in numerous situations as shown in Figures 6, 7, 8, and boxplots are obtained as depicted in Figures 9, 10 and 11. The following evaluations are conducted on eMCOS.

Data transfer latencies between IOS and CC are not influenced by routing. This involved preparing two interfaces (Tx and UC), three routes(direct link, cross link, and detour route), and two memory locations in which the transferred data is allocated. As shown in Figures 6, 7, 8, and 9, 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 torus topology NoC because the number of minimum steps exceeds those in the mesh topology.

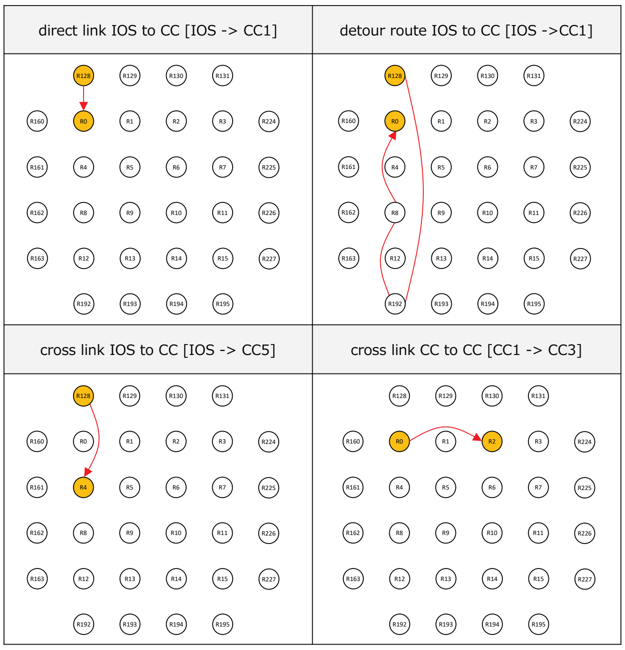

direct link IOS to CC [IOS -> CC1]
detour route IOS to CC [IOS ->CC1]
cross link IOS to CC [IOS -> CC5]
cross link CC to CC [CC1 -> CC3]

Figure 5: Four D-NoC routes used in the evaluation.

It is observed that queuing in the NoC routers and hardware distance on the NoC are not dominant factors for latency. The time taken by transmitting and receiving transactions exceeds those of other transaction. Additionally, it is briefly recognized that the speed of UC exceeds that of Tx. The data is arranged as shown in Figures 10 and 11 to facilitate a precise analysis with respect to the interface and memory location. In the figures, only the cross link from IOS to CC5 is accepted because routes do not influence latency. In order to facilitate intuitive recognition, two kinds of figures are arranged, namely a logarithmic axis and a linear axis.

In the Tx interface, DDR causes a large increase in latency. The time taken by the DDR is twice as that of the IO SMEM as shown in Figure 11. This is due to the memory access speed characteristics of DRAM and SRAM. In the case of the Tx interface, it is necessary for an RM (k1-core) on IOS to operate the DMA in the IOS network 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 IOS and 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., the DDR or SMEM). Similar latency characteristics are observed in Figures 10 and 11. In the case of the UC interface, an RM (k1-core) on the IOS does not involve a DMA transaction. A micro core in the network interface executes a programed 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 the Tx interface, the speed of the data transfer latency between CCs exceeds that between IOS and CC.

4.2 Matrix Calculation

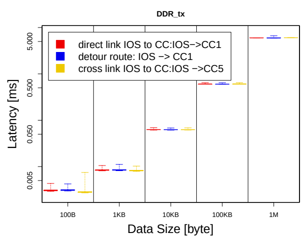


Figure 6: Data transfer with Tx from IO DDR to CC.

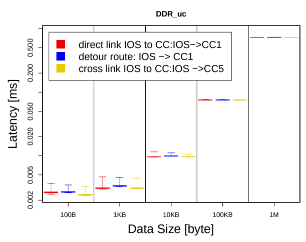


Figure 7: Data transfer with UC from IO DDR to CC.

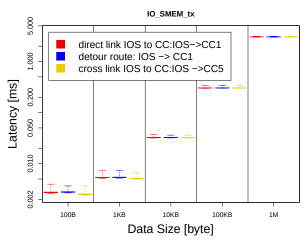


Figure 8: Data transfer with Tx from IO SMEM to CC.

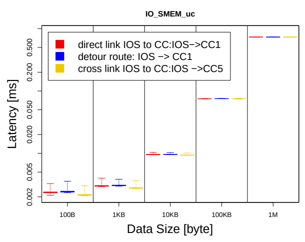


Figure 9: Data transfer with UC from IO SMEM to CC.

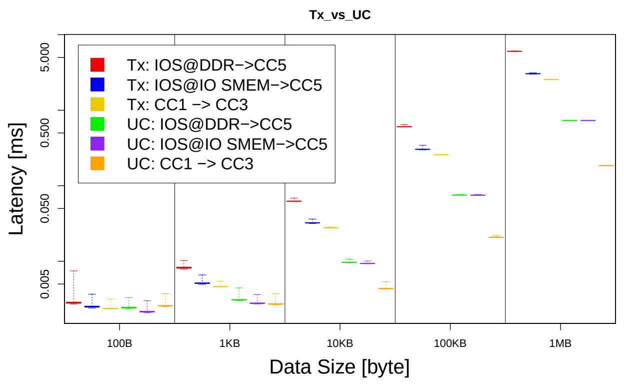


Figure 10: Data transfer with Tx/UC (logarithmic axis).

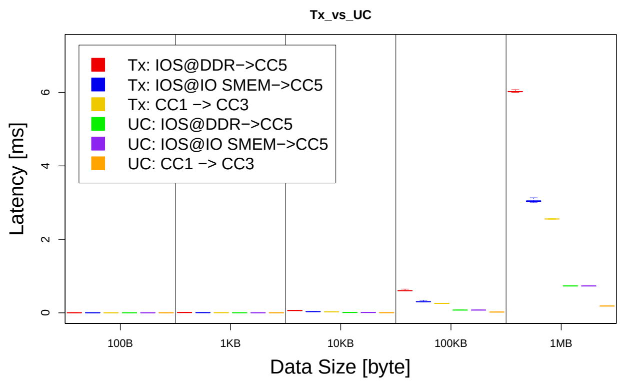


Figure 11: Data transfer with Tx/UC (linear axis).

4.2.1 *Situations and Assumptions*

In the evaluation, matrix calculation time and parallelization potential of MPPA-256 are clarified. Matrix calculations are conducted in IOS and CC. Three computing situations are considered as shown in Figure 12. The first situation involves computing in IOS where four cores are available. In order to analyze memory access characteristics, a matrix buffer is allocated in IO DDR and SMEM. The second situation involves computing in CC in which 16 cores are available. The third situation involves offload-computing by using an IOS and four CCs. Parallelized processing is executed with four CCs' cores and SMEMs. A few cores in IOS and CC manage the parallelized transaction. The method can handle large data in which one cluster is not sufficient because buffer capacity is not limited to 2MB in SMEM. Parallelized processing and the total capacity of SMEM are superior to 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 SMEM. Thus, it is necessary for IOS and CC cores to access matrix buffers without cache to avoid cache coherency trouble. In order to facilitate faster data transfer, a part of the matrix buffer is transmitted in parallel as shown in Figure 12.

Matrix calculation time is analyzed with parallelization and memory allocation. Additionally, the influence of cache is analyzed because cache coherency is an important issue in many-core system. There are several cases in which applications must access specific memory space without a cache. With respect to the given assumptions, maximum total buffer size corresponds to 1 MB, and thus three matrices buffers are prepared, and each size corresponds to 314 KB. Matrix A and matrix B are multiplied, and the result is stored in matrix C. The total of the three matrices is set

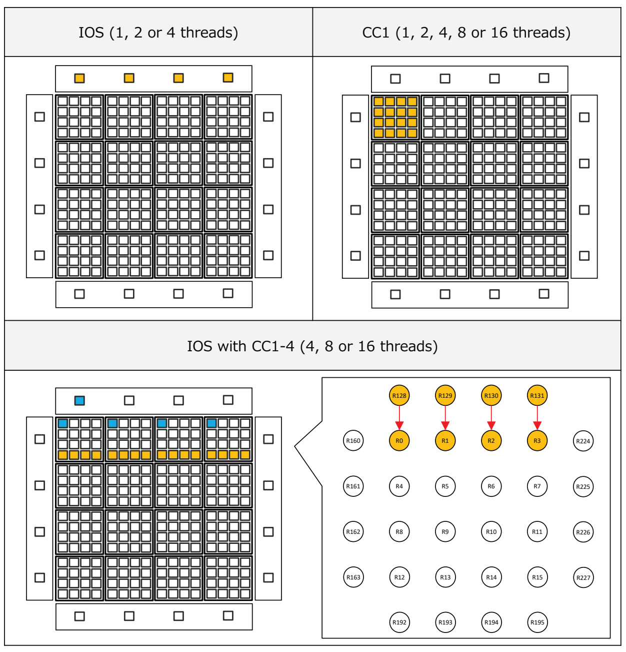
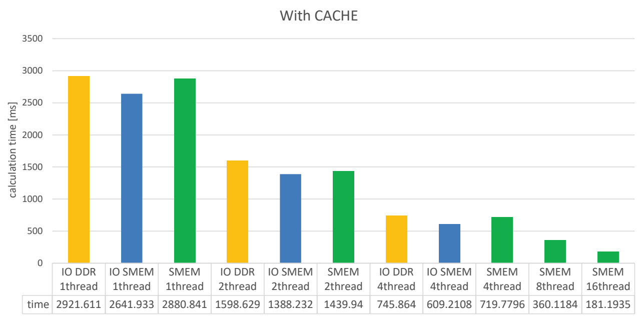


Figure 12: Matrix calculation situations.

as approximately 1 MB. In CC, we assume that the remainder of SMEM (1 MB) is occupied with system software and applications. The following evaluations are conducted on eMCOS.

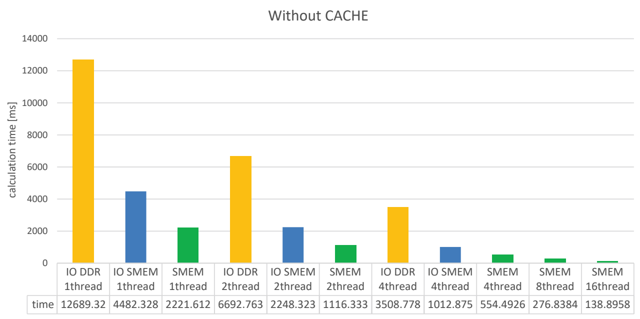
4.2.2 *Influence of Cache and Memory*

First, matrix calculation time with the cache in IOS and



	IO DDR 1thread	IO SMEM 1thread	SMEM 1thread	IO DDR 2thread	IO SMEM 2thread	SMEM 2thread	IO DDR 4thread	IO SMEM 4thread	SMEM 4thread	SMEM 8thread	SMEM 16thread
time	2921.611	2641.933	2880.841	1598.629	1388.232	1439.94	745.864	609.2108	719.7796	360.1184	181.1935

Figure 13: Matrix calculations in IOS and CC with cache.



	IO DDR 1thread	IO SMEM 1thread	SMEM 1thread	IO DDR 2thread	IO SMEM 2thread	SMEM 2thread	IO DDR 4thread	IO SMEM 4thread	SMEM 4thread	SMEM 8thread	SMEM 16thread
time	12689.32	4482.328	2221.612	6692.763	2248.323	1116.333	3508.778	1012.875	554.4926	276.8384	138.8958

Figure 14: Matrix calculations in IOS and CC without cache.

CC is depicted in Figure 13. There are almost no differences between IO DDR, IO SMEM, and CC SMEM due to the cache. Furthermore, 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 cache in the IOS and CC is shown in Figure 14. The absence of a cache, results in a fourfold increase in the DDR and a high difference arises with respect to the SMEM. Another notable result is that calculation speed in CC SMEM exceeds that of the IO SMEM. This characteristic is hidden in the calculation with the cache. The computing cores involve the same k1-cores in IOS and CC, and thus it is considered that the characteristics and physical arrangement of SMEM exert a significant effect. This is an interesting result since a high difference that cannot be ignored exists. It is also observed that calculation time exhibits a linear relation with the number of threads. Furthermore, the calculation speed without the cache in CC SMEM exceeds that with cache. This result is contrary to intuition, and the two factors are conceivable. A small data cache (8 KB) in CC does not function adequately and direct assembly instruction for uncached access optimizes memory access.

4.2.3 *Four CCs Parallelization*

Finally, matrix calculation with offload-computing in IOS and CCs is shown in Figure 15. 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 IO DDR (cached) due to the fore-mentioned assumption. An overhead transactio (inverse matB and input matC) is performed to offload a part of calculation on the four CCs and aggregate calculation result. This produces a overhead irrespective of the number of threads as shown in Figure 15. However, the speed involved in offloading result exceeds that of 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 RM (k1-core)'s memory access. In the offloading case, a DMA accesses matrix buffers on DDR and transfers the buffers from IO DDR to each CC SMEM. Subsequentl, PEs in the CC access matrix buffer the calculation without 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 size is large as shown in Figure 16. Only a part of matrix is allocated in CC, and thus it is possible to handle larger matrices buffers. Additionally, 640 KB matrices are prepared, and matrix calculation calculations are evaluated with offload-computing. The speed of the offloading result exceeds that of IO DDR result with respect to the 314 KB matrices in Figure 15.

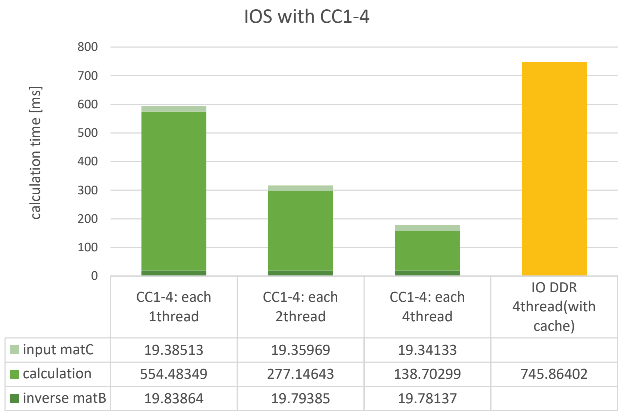
4.3 Case Study

The application in this case study is a part of autonomous driving systems and this section demonstrates the parallelization potential of the MPPA-256. We port a algorithm of vehicle self-localization written in C++ in Autoware, namely open-source software for urban autonomous driving [1], [7], and parallelize a part of it. The self-localization adopts normal-distributions transform matching algorithm [21] implemented in Point Cloud Library [2].

The self-localization algorithm used in this case study is mainly divided into two processes; *RadiusSearch* that searches for several nearest neighbor points for each query and calculates the distance, *Derivatives* in which the derivative is calculated necessary to determine the convergence of the matching operation. This evaluation parallelize *Derivatives* in 16 CCs and the remainder of the algorithm is executed on IOS. In order to parallelize *RadiusSearch*, the algorithm of nearest neighbor search has to be redesigned necessary since the data to be searched exceeds 1 MB. Redesigning the algorithm of nearest neighbor search is future work for this study. Figure 17 shows the average execution time for each convergence and clarifies that the parallelization accelerates the process of *Derivatives*. This acceleration clarifies parallelization potential of the MPPA-256. Since the reduction of execution time of *derivatives* involves reducing the number of the loop for convergence, execution times of *RadiusSearch* and other parts are also shortened. The query can be assumed to be 10 Hz in many automated driving systems. Hence, if we succeed the parallelization of *RadiusSearch*, the goal will be able to achieve.

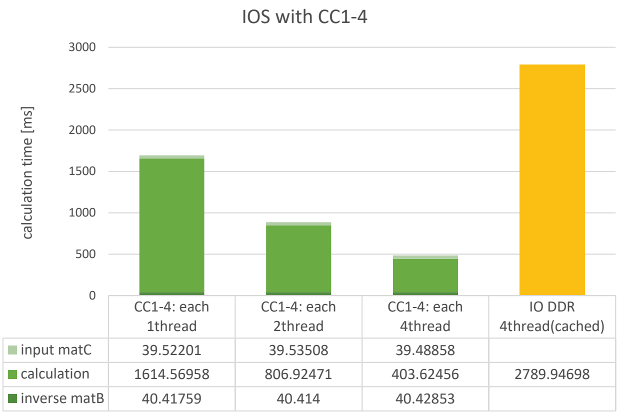
5. RELATED WORK

This section compares many-core platforms and discusses previous studies related to multi/many cores. First, needs of high-performance computation and comparison of many-core platforms to other platforms are discussed. Second, the Kalray MPPA-256 which this study focuses on is compared to other COTS COTS many/multicore components, and we summarize the features of MPPA-256. Finally, discussions



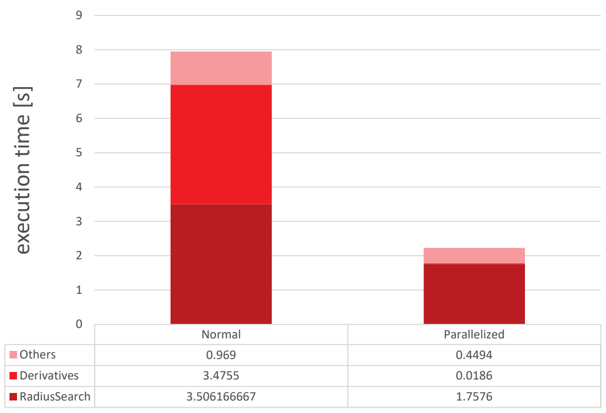
	CC1-4: each 1thread	CC1-4: each 2thread	CC1-4: each 4thread	IO DDR 4thread(with cache)
🟩 input matC	19.38513	19.35969	19.34133	
🟩 calculation	554.48349	277.14643	138.70299	745.86402
🟩 inverse matB	19.83864	19.79385	19.78137	

Figure 15: Matrix calculations with offload-computing (314 KB matrix x 3).



	CC1-4: each 1thread	CC1-4: each 2thread	CC1-4: each 4thread	IO DDR 4thread(cached)
🟩 input matC	39.52201	39.53508	39.48858	
🟩 calculation	1614.56958	806.92471	403.62456	2789.94698
🟩 inverse matB	40.41759	40.414	40.42853	

Figure 16: Matrix calculations with offload-computing (640 KB matrix x 3).



	Normal	Parallelized
🟥 Others	0.969	0.4494
🟥 Derivatives	3.4755	0.0186
🟥 RadiusSearch	3.506166667	1.7576

Figure 17: Vehicle self-localization of autonomous driving application with a part of parallelization.

of previous studies and comparison to them are described.

Recently, studies indicate that the single core processors are characterized by limited computation performance. Pollack stated that a single core is inefficient [27] and thar Moore's law [23] is no longer applicable. Therefore, extant CPUs are not sufficient to satisfy increasing computation demands. The shift from a single core in real-Time and embedded systems has occurred [30]. Many other platforms including many-core are developed and researched by current studies.

Table 1 summarizes the features of many-core platforms with that of other platforms. For instance, the GPU is a powerful device to enhance computing performance and it has great potential in specific areas (for e.g., image processing, and learning). However, it is mainly used for a specific purpose and its reliability is not suitable for real-time systems. It is difficult to use a GPU for a global purpose and to guarantee its reliability due to the GPU architecture. Many-core is significantly superior to GPU with respect to a global purpose and multiple instructions. Additionally, it is commonly known that many-core involves a reasonable power consumption. In contrast, the GPU consumes a significant amount of power and generates considerable heat. This is a critical problem for embedded systems. Furthermore, DSPs and FPGAs are also high-performance devices when compared to CPUs. They are efficient in terms of power consumption. A DSP is often used for real-time systems and FPGAs guarantee reliability and efficient processing. They are suitable for time-critical computing [12]. However, DSPs cannot be used for global purpose and programing. Furthermore, FPGAs are difficult for software developers since their software model is ignificantly different from that of CPU and is not a substitute for CPU. Many-core platforms can potentially replace single/multi core CPU as they possess ease of programing and scalability with high acceleration.

Based on the fore-mentioned background, many COTS many/multicore components are developed and released by several vendors. (e.g., Kalray's the Multi-Purpose Processing Array (MPPA) 256 [14], [13], [17], [3], Tilera's Tile-Gx [31], [29], Tilera's Tile64 [6], and Intel's Xeon Phi [9], [10], Intel's Single-chip Cloud Computer (SCC) [3]). The present study focuses on the Kalray MPPA-256 that is designed for real-time embedded applications. Kalray S.A. [14], [13], [17] presented clustered many-core architectures on the NoC which pack 256 general-purpose cores with high energy efficiency.

MPPA-256 is superior to other COTS many/multicore components in terms of the scalability of the number of cores and power efficiency as shown in Table 2. In terms of the scalability, MPPA-256 adopts 256 cores while other COTS many/multicore components pack 64 cores or the number of cores around it. This scalability of cores is attributed to the memory architecture that each cluster per 16 cores contains its own local shared memory. The precise hardware model is described in Section 2.1. When all cores share global DDR memory like other platforms besides MPPA-256, specific bus route receives extremely much load and memory access contention frequently occurs. Local shared memories reduce above problems and help the scalability of the number of cores. This is the reason why MPPA-256 has succeeded in scaling the number of cores up to 256 cores. However, the local shared memory architecture restricts the capacity of memory and needs a data copy from DDR with NoC. This restriction makes use of extant applications difficult especially in the case of applications which require more memory. In terms of power efficiency, MPPA-256 realizes superior energy efficiency despite a large number of cores. Total clock frequency per watt is the highest in the current

Table 1: Comparison of Many-core to CPU, GPU, DSP, and FPGA

	performance	power/heat	reliability	real-time	software development	costs	multiple instruction
CPU		△	✓	✓	✓	✓	△
GPU	✓		△		△	✓	
DSP	△	△	✓	✓	△	✓	
FPGA	✓	△	✓	△		△	
Many-core	✓	✓	✓	✓	✓	△	✓

Table 2: Comparison of Many-core Processors Platforms

	scalability	power efficiency	code transplant
Kalray MPPA-256 [14], [13], [17]	✓	✓	△
Tilera Tile series [6]		△	✓
Intel Xeon Phi [9] [10]			✓
Intel SCC [3]			✓

COTS many/multicore components. While MPPA-256 and TILE64 [6] target embedded systems, other COTS components consider HPC accelerator. Their clock frequency per core exceeds embedded solutions' one. We need to distinguish the COTS many/multicore components according to requirements with reference to Table 2. MPPA-256 is typically accepted with respect to many-core platforms and the model are used in several previous studies [26], [4], [8], [25], [24].

Several extant studies examined real-time applications on many-core platforms including MPPA-256. In [30], many opportunities and challenges of many/multicore platforms are discussed. The shift to many/multicore in real-Time and embedded systems is described.

Based on the above background, several taks mapping algorithms for many/multicore systems are proposed [25], [28], [8], [5], [18]. Airbus S.A.S. [25] proposes the method of Directed Acyclic Graph (DAG) scheduling for hard real-time applications using MPPA-256. In [18], mapping framework is proposed based on AUTOSAR which is applied as a standard architecture to develop automotive embedded software systems [19]. AUTOSAR tasks scheduling considering contention in shared resources is presented in [4].

Examining the above mapping algorithms of real-time applications, several previous studies [20], [16], [15], [24] analyze the potential of MPPA-256 and data transfer with NoC as shown in Table 3. First, MPPA-256 is introduced, and its performance and energy consumption are reported in [20]. However, the report contains few evaluations and does not refer to data transfer with NoC and memory access characteristics. Second, data transfer with NoC in MPPA-256 is described, and NoC guaranteed services is analyzed in [16] and [15]. While theoretical analysis is done thoroughly, practical evaluations are poor and parallel data transfer does not be referred. Finally, the authors of [24] focus on predictable composition of memory accesses. Analysis of this study identifies the external DDR and the NoC to be the main bottlenecks for both average performance and predictability in platforms like MPPA-256. Though the analysis clarifies memory access characteristics of the external DDR and provides notable lessons, a solution for DDR bottleneck is not examined and practical evaluations are lacked.

6. CONCLUSIONS

This paper has conducted quantitative evaluations which clarify end-to-end latencies with many-core computing based on NoC, the scalability of parallelization, and the characteristics of memory access speed. From various experiments, we have evaluated the capabilities of the currently available data transfer methods and parallelization potential for many-core computing base on NoC. Our experimental results allow system designers to choose appropriate system design based on the requirement of their latency-sensitive many-core applications beyond an intuitive expectation.

In future work, we will port real-time systems software such as [22] and propose the parallelization of memory intensive algorithms such as nearest neighbor search in Section 4.3.

7. ACKNOWLEDGEMENTS

We acknowledge the support by eSOL and Kalray. The authors like to thank Atsushi Matsuo and Masaki Gondo from eSOL, and Gilbert Richard and StÃľphane Cordova from Kalray for their support which helped to improve this work.

8. REFERENCES

Table 3: Comparison of Related Work

	performance analysis	data transfer analysis with NoC	memory access characteristics	real applications	parallel data transfer
Kalray clusters calculate quickly [20]	△				
Network-on-Chip Service Guarantees [16]		✓			
Predictable composition of memory accesses [24]		✓	✓		
this paper	✓	✓	✓	✓	✓