Exploring Scalable Data Allocation and Parallel Computing on NoC-based Embedded Many Cores

Abstract—**In embedded systems, high processing requirements and low power consumption need heterogeneous computing platforms. Considering embedded requirements, applications need to be designed based on scalable data allocation and parallel computing with non-uniform memory access (NUMA) many cores. In this paper, we use one of the embedded commercial off-the-shelf (COTS) multi/many-core components, the Massively Parallel Processor Arrays (MPPA) 256 developed by Kalray, and conduct evaluations of data transfer and parallelization micro benchmark. We investigate currently achievable data transfer latencies between distributed memories on networks-on-chip (NoC), memory access characteristics, and parallelization potential with many cores. Subsequently, we run a practical application, the core of the autonomous driving system, on many-core processors and acceleration by parallelization indicates practicality of many cores. By highlighting many-core computing capabilities, we explore the scalable data allocation and parallel computing on NoC-based embedded many cores.**

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 in several domains such as automobiles. In embedded systems, this trend is also adapted because of high processing requirements. For example, an autonomous driving system involves various applications and are sometimes characterized by demands for high-performance computing. Considering requirements of high processing, predictability, and energy efficiency for autonomous driving systems, its system needs a heterogeneous computing system such as multi/many cores and GPUs. (This is discussed in Section 4.) In embedded systems, Multi/many-core architecture is an important trend as it integrates cores to realize high-performance and general-purpose computing with low power consumption [4], [13], [19].

Despite the emergence of the need for multi/many-core platforms, several difficulties persist in the adaptation of these platforms to embedded systems [4], [22]. These difficulties are caused by strict requirements of embedded systems and the hardware architecture shares resources (e.g. memory subsystems and I/O devices) by many cores. Parallelized processes share memories, which leads to the frequent occurrence of conflicts. Shared resources also disturb predictable timing behavior and software analysis. Although it is possible to connect a large memory and all the cores with wide-bandwidth buses, it will lose its scaleability due to bus competition and it will require a large power consumption. For instance, the Multi-Purpose Processing Array (MPPA) 256 developed by Kalray [12] adopt non-uniform memory access (NUMA) using network-on-chip (NoC), and realize numerous cores and low power consumption.

Considering embedded requirements, multi/many cores need NUMA because of scalability of the number of cores and reasonable power consumption. Scalable data allocation enhances parallelized high-performance and general-purpose computing with low power consumption. MPPA-256 is one of the commercial off-the-shelf (COTS) multi/many-core components targeting embedded system and adopts distributed memory architecture with NoC. (We discusses comparison to other platforms in Section 4.) However, its data transfer between distributed memories with NoC, parallelization potential, and memory access characteristics are not fully unrvealed for application developers. This work explores above issues with quantitative evaluations and a practical application.

Contributions: This work focuses on examining embedded many-core computing based on NoCs, such as MPPA-256. We conduct evaluations of data transfer methods on NoCs and micro benchmark with matrix calculation to clarify latency characteristics of data transfer, parallel computing, and an influence of data allocation. Subsequently, we parallelize localization algorithm which is the core of the autonomous driving system. We reveal the advantages and disadvantages of embedded many-core computing based on NoC in a quantitative way by leading the following contributions:

- The evaluations of the data transfer on DMA-capable NoC 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 is observed in evaluations of matrix calculations in several situations and a real complex application as a part of an autonomous driving system.
- The evaluations of the parallel processing examine the characteristics of the memory access speed which varies according to where data is 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 autonomous

driving application indicates a practical potential for NoC-based embedded many-core computing.

Organization: The remainder of this work is organized as follows. First, 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. Second, Section 3 explains evaluation setup and approach, and illustrates experimental evaluations. Subsequently, Section 4 examines related work that focus on multi/many-core systems. Finally, Section 5 presents the conclusions and directions for future research.

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 2.1, 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 and 2). The MPPA MANYCORE chip integrates 16 compute clusters and 4 IOSs 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 (IOSs): North, South, East, and West IOSs. 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 10Gb/s Ethernet controller. Two pairs of IOSs organize two I/O clusters (IOCs) as shown in Figure 1. Each IOS comprises of quad IO cores and a NoC interface.

IO cores: IO cores are connected to a 16-banked parallel shared memory with a total capacity (IO SMEM) of 2 MB, as shown in Figure 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 the PCIe, Ethernet, and other I/O devices. They operate the local peripherals, including the NoC interfaces with DMA.

NoC Interface: The NoC interface contains four DMA engines (DMA1-4) and four NoC routers as shown in 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 and it has the following three NoC interfaces: a receive (Rx) interface, a transmit (Tx) interface, and a micro core (UC). A UC is a fine-grained multi-threaded engine 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 Element (PE) and an IO core.

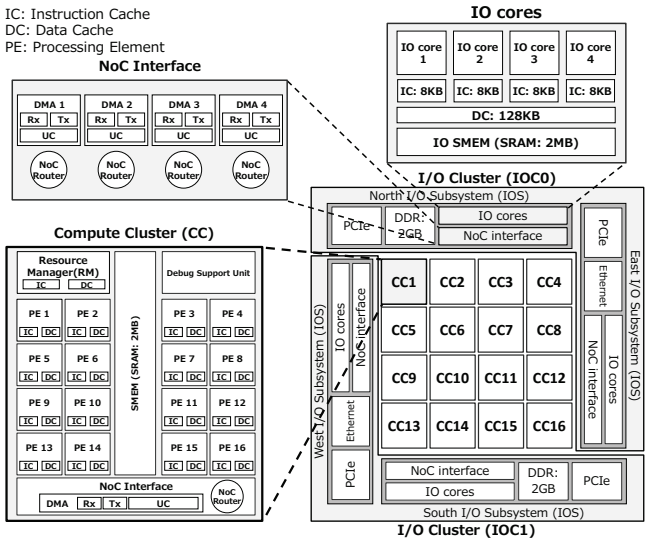


Figure 1. An overview of the architecture of the Kalray MPPA-256 Bostan.

2.1.2. Compute Clusters (CC). In MPPA-256, the 16 inner nodes of the NoC correspond to the CCs. Figure 1 illustrates the architecture of each CC.

Processing Elements and an RM: In a CC, 16 processing elements (PEs) and an RM share 2 MB cluster local memory (SMEM). 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 600 or 800 MHz. Each core is fitted with its own instruction and data caches.

A Debug Support Unit and a NoC Interface: In addition to PEs and an RM, bus masters on 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.

2.1.3. Network-on-Chip (NoC). The 16 CCs and the 4 IOSs are connected by a NoC as shown in Figure 2. Furthermore, a NoC is constructed as the bus network and has routers on each node.

Bus Network: Bus network connects nodes (CCs and IOSs) with torus topology [9] which involves a low average number of hops when compared to mesh topology [25], [24]. The network is actually composed of the following two parallel NoCs with bi-directional links (denoted by red lines in Figure 2): 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. Data is 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 hold the following two routers of its own: a D-NoC router and a C-NoC router Each RM or IO core on NoC node is associated with the fore-mentioned

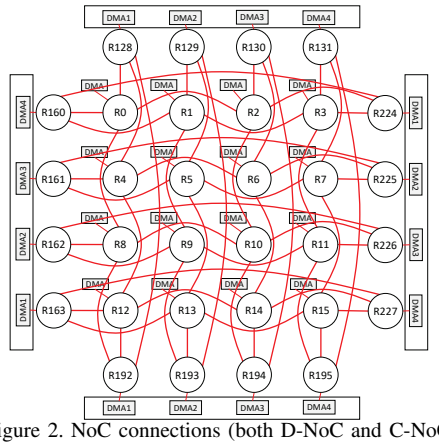


Figure 2. NoC connections (both D-NoC and C-NoC).

two 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. The NoC routers shown in Figure 1 ared illustrated as nodes in Figure 2. 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 3 shows the software stack used for Kalray MPPA-256 in the present work. With respect to scalable computing array of the system, it is possible to map several programming models or runtimes such Linux, 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, 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. 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 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 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. Librouting offers a minimal set of functions that can be used to route data between any clusters of the MPPA.

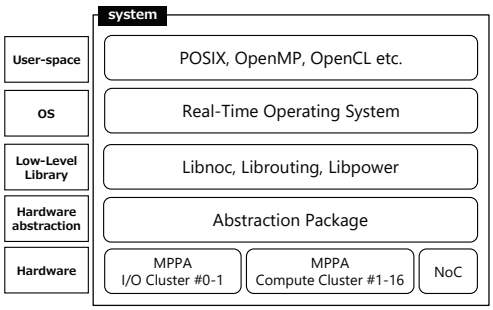


Figure 3. The software stack of Kalray MPPA-256.

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. RTEMS runs on the IOC except for the CC.

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.

eMCOS: On both CC and IOS, eMCOS provides minimal programming 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. It enables applications to operate priority based message passing, local thread scheduling, and thread management on IOS as well as CC.

3. Evaluations

At First, 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. Subsequently, we conduct practical autonomous driving application to examine practicality of NUMA many cores. Finally, we arrange lessons learned from above evaluations. The following all evaluations are conducted on real hardware board with eMCOS.

3.1. D-NoC Data Transfer

3.1.1. 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 Figure 4. The routes on D-NoC map (Figure 2) contains various connections between routers, namely a direct link, a cross-link, and a flying link. With respect to the case of routes from the IOS routers

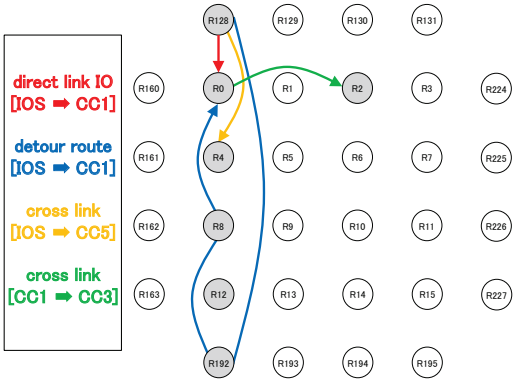


Figure 4. Four D-NoC routes used in the evaluation.

to the CC routers, transmitted data is allocated in DDR or IO SMEM. The CC only includes SMEM as shown in Figure 1. 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 is 2 MB, and thus it is assumed that the appropriate communication buffer size is 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 5, 6, 7, and boxplots are obtained as depicted in Figures 8 and 9.

3.1.2. Data Transfer Methods. In this section, data transfer methods in MPPA-256 are explained. For scalability purposes, MPPA-256 accepts a clustered architecture which avoids frequent memory contention between numerous cores. However, the architecture constraints memory that 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 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 it to wait to receive the data. A DMA in a NoC 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. 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.

3.1.3. Influences of Routing and Memory Type. Data transfer latencies between IOS and CC are not hardly 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 5, 6, 7, and 8, end-to-end latency scales exhibit a linear relation with data size, and

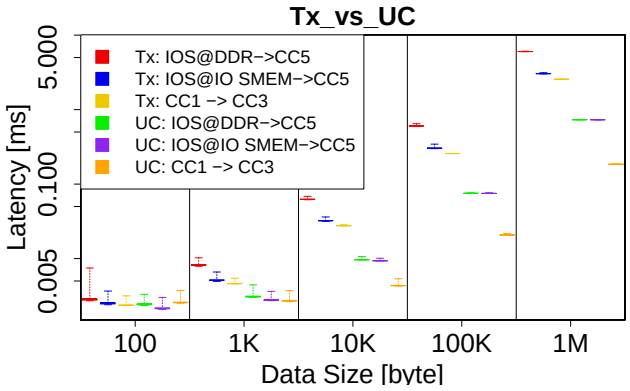


Figure 9. Data transfer with Tx/UC

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. It is observed that queuing in the NoC routers and hardware distance on the NoC are not dominant factors for latency. The router latency, the time taken by transmitting and receiving transactions in a RM, 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 Figure 9 to facilitate a precise analysis with respect to the interface and memory location. In Figure 9, only the cross-link from IOS to CC5 is accepted because routes do not influence latency.

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 9. 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 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 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). 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 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.

3.2. Matrix Calculation

3.2.1. 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 10. 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

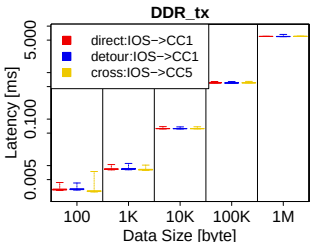


Figure 5. Data transfer with Tx from IO DDR to CC.

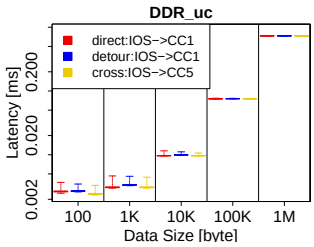


Figure 6. Data transfer with UC from IO DDR to CC.

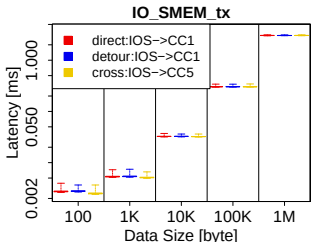


Figure 7. Data transfer with Tx from IO SMEM to CC.

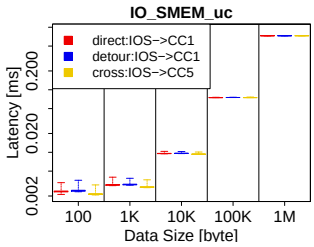


Figure 8. Data transfer with UC from IO SMEM to CC.

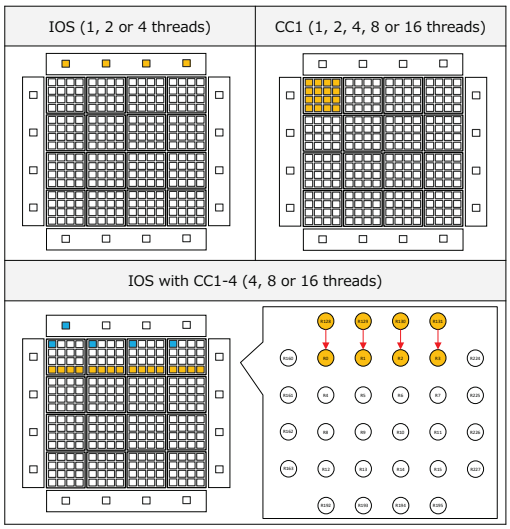


Figure 10. Matrix calculation situations.

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 2 MB in SMEM. 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 10.

Matrix calculation time is analyzed with parallelization and memory allocation. Additionally, the influences of cache is 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. With respect to the given assumptions, maximum total buffer size is 1 MB, and thus three matrices buffers are prepared, and each size is 314 KB. Matrix A and matrix B are multiplied, and the result is stored in matrix C. We assume that the remainder of SMEM (1 MB) is occupied with system software and applications in CC.

3.2.2. Influences of Cache and Memory Type. First, matrix calculation time with the cache in IOS and CC is depicted in Figure 11. There are almost no differences between IO DDR, IO SMEM, and CC SMEM due to the cache. 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 Figure 12. 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 physically involvethe same 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 a cache line problem are conceivable. When a small data cache (8 KB) in a PE of CC does not function adequately and an application always misses cache, memory access will pay the time for a non-cached data access and the cost to refill the cache line. As a result, the memory access speed without the cache exceeds that with cache.

3.2.3. Four CCs Parallelization. Finally, matrix calculation with offload-computing in IOS and CCs is shown in Figure 13. 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. The aggregate calculation is obtained by offloading on the four CC to perform a multiplication of a tile of matA and a tile of the transpose of matB. This produces an overhead irrespective of the number of threads as shown in Figures 13 and 14. 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 IO core's memory access even if target memory is allocated on DDR. In the offloading case, a DMA accesses matrix buffers on DDR and transfers the buffers from IO DDR to each CC SMEM. Subsequently, 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

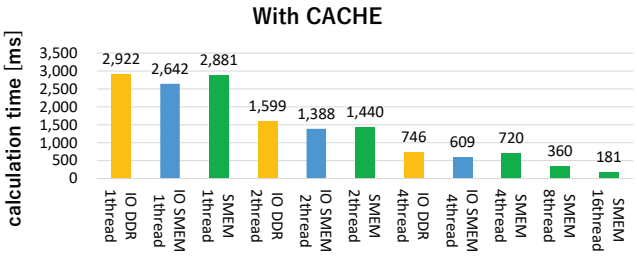


Figure 11. Matrix calculations in IOS and CC with cache.

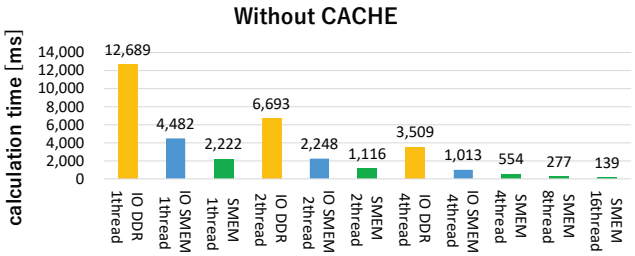
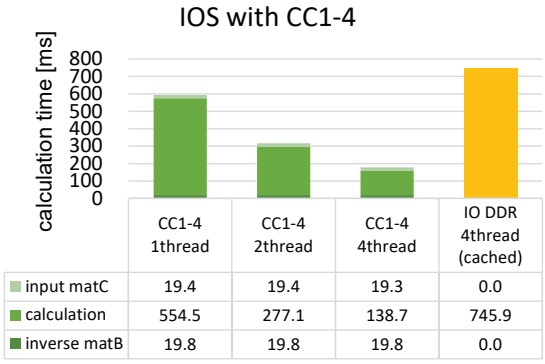


Figure 12. Matrix calculations in IOS and CC without cache.



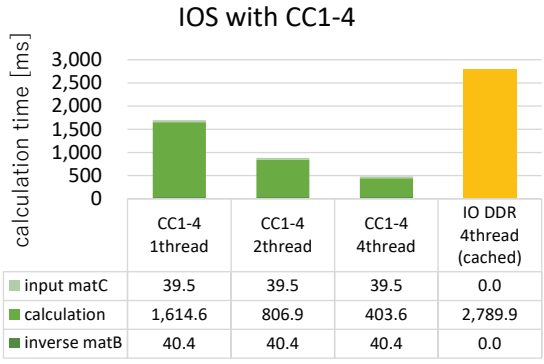


Figure 13. Matrix calculations with offload-computing (314 KB matrix x 3). Figure 14. Matrix calculations with offload-computing (640 KB matrix x 3).

increases when the matrix size is large as shown in Figure 14. 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 DDR, NoC router's FIFOs sometimes overflow and cause an error due to memory access delay of DDR. This error should be prevented by transmission protocol, and flow control is future work for MPPA-256. Currently, to avoid this error, matrix C should be allocated in IO SMEM. Note that above evaluation results when matrix C is allocated in DDR.

3.3. Practical Application

This wotk adopts a part of an autonomous driving system and this section demonstrates the parallelization potential of the MPPA-256. We took an algorithm for vehicle self-localization written in C++ in Autoware, namely an open-source software for urban autonomous driving [1], [16], and parallelized part of it. The self-localization adopts the normal-distribution transform matching algorithm [17] implemented in the Point Cloud Library [2].

The self-localization algorithm is primarily divided into two processes: *RadiusSearch* which searches for several nearest neighbor points for each query and calculates the distance, and *Derivatives* which calculates the derivative to determine the convergence of the matching operation. This evaluation parallelized *Derivatives* onto 16 CCs and the remainder of the algorithm was executed on the IOS with its four cores. To parallelize *RadiusSearch*, the algorithm of the nearest neighbor search needs to be redesigned because the data to be searched exceeds 1 MB. Redesigning this algorithm will be part of a future work.

As shown in Figure 15, the evaluation of parallelized self-localization algorithm indicates the average execution time for each convergence and demonstrates that the parallelization accelerates the *Derivatives* process. This acceleration shows the parallelization potential of the MPPA-256. Because the reduction of the execution time of *Derivatives* involves reducing the number of loops for convergence, the execution times of *RadiusSearch* and other parts are also shortened. The query can be assumed to be 10 Hz in many automated driving systems. Therefore, if we succeed in the parallelization of *RadiusSearch*, this goal can be achieved.

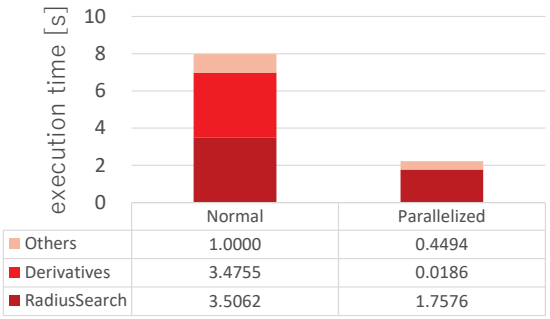


Figure 15. Vehicle self-localization of the partially parallelized autonomous driving application.

3.4. Lessons Learned

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

From evaluations of D-NoC data transfer, we can get two lessons: influences of NoC routing and DMA. First, data transfer latencies between Clusters are hardly influenced by routing for users as shown in Figures 5, 6, 7, and 8. Software transactions of transmitting and receiving in routers and RM are dominant factors for latencies. Second, in the IOS, the latency of UC is not significantly affected by the memory

location, the DDR or SMEM, at which the transferred buffer is allocated from the evaluation of Figure 9. The merit of UC is profitable for users because it is common situations to transfer data from DDR to CC SMEM using UC, especially for large data.

From micro benchmark of matrix calculation, we can get three lessons of memory access: characteristic of SMEM, influences of cache, and data flow control on D-NoC. First, memory access speed of CC SMEM exceeds that of the IO SMEM as shown in Figure 12. There is a significant difference, which is a major reason to actively handle in CC. Second, we quantitatively indicate that the calculation speed without the cache in CC SMEM in Figure 12 exceeds that with cache in Figure 11. It is generally known that cache overhead exists when the miss hit frequently occurs, but it is notable that there is a big difference which can not be ignored by the influences of cache line. Third, when data is parallelly transferred from multiple CCs to IO DDR, NoC routers' FIFO sometimes overflows due to memory access delay of DDR. This should be prevented by transmission protocol and flow control.

From the practical application, since we parallelize the vehicle self-localization algorithm of the autonomous driving system, it can be practically used. We apply for parallel data transferfrom IOS to CCs and parallel computing on IOS and CC by using CC SMEM as scratch pad memory.

4. Related Work

This section compares many-core platforms and discusses previous work related to multi/many cores. 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 COTS multi/many-core components, and we summarize the features of MPPA-256. Finally, discussions of previous work and comparison to them are described.

Table 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 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 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 [15]. 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

TABLE 1. COMPARISON OF MANY-CORE TO CPU, GPU, AND FPGA

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

∗In a table, "L" means "limited".

TABLE 2. COMPARISON OF MANY-CORE PROCESSOR PLATFORMS

	scalability	power efficiency	code transplant
Kalray MPPA-256 [12]	✓	✓	L
Tilera Tile series [5]		L	✓
Intel Xeon Phi [7] [8]			✓
Intel SCC [3]			✓

single/multi core CPU as they possess ease of programming and scalability.

Based on the fore-mentioned background, many COTS multi/many-core components are developed and released by several vendors. (e.g., Kalray's the Multi-Purpose Processing Array (MPPA) 256, [12], Tilera's Tile-Gx [21], [23], Tilera's Tile64 [5], and Intel's Xeon Phi [7], [8], Intel's Single-chip Cloud Computer (SCC) [3]). The present work focuses on the Kalray MPPA-256 that is designed for real-time embedded applications. Kalray, [12] 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 multi/many-core components in terms of the scalability of the number of cores and the power efficiency as shown in Table 2. In terms of the scalability, MPPA-256 adopts 256 cores while 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 2.1. When all cores share the global DDR memory as in other platforms excluding MPPA-256, specific bus routes receives 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. This is why MPPA-256 has succeeded in scaling up to 256 cores and what the "scalability" column of Table 2 means. However, the NUMA memory architecture restricts the capacity of the memory and requires a data copy from 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, due to NUMA memory architecture, portability of code porting to MPPA-256 is inferior to other COTS platforms as shown in Table 2.

In terms of power efficiency, MPPA-256 realizes superior energy efficiency despite its large number of cores [15]. 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 16W at 600 MHz and 24W at 800 MHz. We need to distinguish the COTS multi/many-core components according to their requirements with reference to Table 2. MPPA-256 is typically accepted with respect to many-core platforms and the model has been used in previous work [4], [6], [19], [20].

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 [15]	L				
Network-on-Chip Service Guarantees [11]		✓			
Predictable composition of memory accesses [20]		✓	✓		
this paper	✓	✓	✓	✓	✓

Previous work have examined real-time applications on many-core platforms including MPPA-256. In Ref. [22], multiple opportunities and challenges of multi/many-core platforms are discussed. The shift to multi/many cores 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 [6], [13], [19]. Airbus [19] proposes a method of directed acyclic graph (DAG) scheduling for hard real-time applications using MPPA-256. In Ref. [13], a mapping framework is proposed on the basis of AUTOSAR which is applied as a standard architecture to develop automotive embedded software systems [14]. AUTOSAR task scheduling considering contention in shared resources is presented in Ref. [4].

By examining the above mapping algorithms of real-time applications, previous work [10], [11], [15], [20] have analyzed the potential of MPPA-256 and data transfer with NoC as shown in Table 3. MPPA-256 is introduced, and its performance and energy consumption are reported in Ref. [15]. 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 Refs. [10] and [11]. While the theoretical analysis is thorough in these work, the practical evaluations are poor and parallel data transfer is not referred to. The authors of Ref. [20] focused on the predictable composition of memory accesses. An analysis of their work identified the external DDR and the NoC as main 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.

5. Conclusions

This work has conducted quantitative evaluations of data transfer methods on NoCs, micro benchmark with matrix calculation and practical application with NUMA many cores such as MPPA-256. Evaluations indicate latency characteristics on NoC, influence of data allocation, and the scalability of parallelization. Our experimental results will allow system designers to choose appropriate system designs. At the last, parallelization of real application proofs practicality of NoC-based embedded NUMA many-cores.

In future work, we will port real-time systems software such as that in Ref. [18] and propose the parallelization of memory intensive algorithms such as the nearest neighbor search in Section 3.3.

References

[1] Autoware: Open-source software for urban autonomous driving. https://github.com/CPFL/Autoware.

[2] Point Cloud Library (PCL). http://pointclouds.org/.

[3] M. Baron. The single-chip cloud computer. *Microprocessor Report*, 24:4, 2010.

[4] M. Becker, D. Dasari, B. Nicolic, B. Akesson, T. Nolte, et al. Contention-free execution of automotive applications on a clustered many-core platform. In *Proc. of 28th IEEE ECRTS*, pages 14–24, 2016.

[5] 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 ISSCC*, pages 88–598. IEEE, 2008.

[6] 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 the 14th IEEE ACSD*, pages 112–121, 2014.

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

[8] G. Chrysos and S. P. Engineer. Intel Xeon Phi coprocessor (codename knights corner). In *Proc. of the 24th Hot Chips Symposium*, 2012.

[9] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. of IEEE DAC*, pages 684–689, 2001.

[10] 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 NoCArc*, pages 11–16, 2014.

[11] B. D. de Dinechin and A. Graillat. Network-on-chip service guarantees on the kalray mppa-256 bostan processor. In *Proc. of the 2nd AISTECS*, 2017.

[12] 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 IEEE DATE*, pages 1–6, 2014.

[13] 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 ETFA*, pages 1–9, 2014.

[14] 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 the 14th ELIV*, volume 62, 2009.

[15] D. Kanter and L. Gwennap. Kalray clusters calculate quickly. *Microprocessor Report*, 2015.

[16] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada. An Open Approach to Autonomous Vehicles. *IEEE Micro*, 35(6):60–68, 2015.

[17] M. Magnusson. *The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection*. PhD thesis, Örebro universitet, 2009.

[18] Y. Maruyama, S. Kato, and T. Azumi. Exploring the Performance of ROS2. In *Proc. of the 13th ACM EMSOFT*, pages 5:1–5:10, 2016.

[19] 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 the ACM 24th RTNS*, pages 235–244, 2016.

[20] 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 the 8th ECRTS*, 2016.

[21] C. Ramey. TILE-Gx100 manycore processor: Acceleration interfaces and architecture. In *Proc. of the 23th IEEE HCS*, pages 1–21. IEEE, 2011.

[22] 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 IEEE CODES+ISSS*, pages 220–229, 2015.

[23] R. Schooler. Tile processors: Many-core for embedded and cloud computing. In *Proc. of HPEC*, 2010.

[24] 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*, 22(2):25–35, 2002.

[25] 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 ISSCC*, pages 98–99, 2007.