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

Yuya Maruyama
Graduate School of Engineering Science
Osaka University

Shinpei Kato
Graduate School of
Information Science and Technology
The University of Tokyo

Takuya Azumi
Graduate School of Engineering Science
Osaka University

***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 of a practical application. We investigate currently achievable data transfer latencies between distributed memories on network-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. 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 [2], [7].

Despite the emergence of the need for multi/many-core platforms, several difficulties persist in the adaptation of these platforms to embedded systems [2], [8]. 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. MPPA-256 [4] is one of the commercial off-the-shelf (COTS) multi/many-core components targeting embedded system. It is developed by Kalray and adopt non-uniform memory access (NUMA) using network-on-chip (NoC), and realize numerous cores and low power consumption. However, its data transfer between distributed memories with NoC, parallelization potential, and memory access characteristics are not fully unrevealed 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 NoC, such as MPPA-256. We conduct evaluations of data transfer methods on NoC to clarify latency characteristics of data transfer and an influence of data allocation. Subsequently, we parallelize a 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 a real complex application as a part of an autonomous driving system.

2. System Model

This section presents the system model used throughout the work. The many-core model of Kalray MPPA-256 Bostan is considered.

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 many-core chip integrates 16 CCs and 4 IOSs on NoC.

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

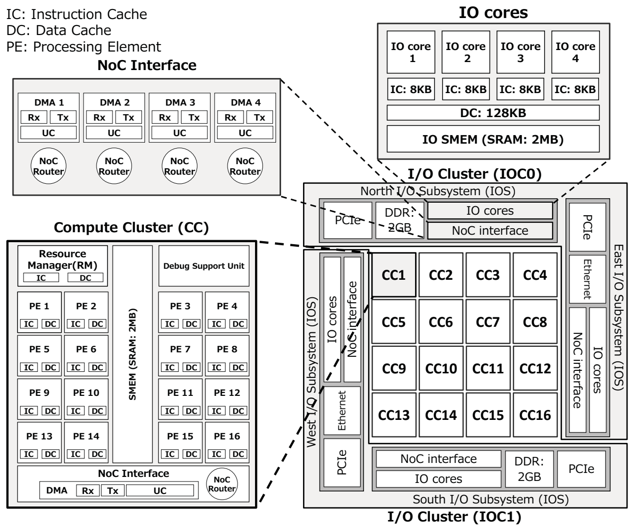


Figure 1. An overview of the architecture of the Kalray MPPA-256 Bostan.

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.

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.

2.1.2. Compute Clusters (CCs).

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.

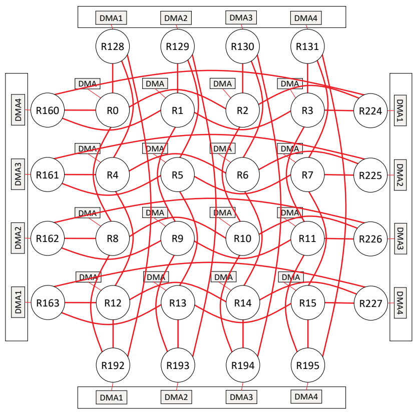


Figure 2. NoC connections (both D-NoC and C-NoC).

2.1.3. Network-on-Chip (NoC).

The 16 CCs and the 4 IOSs are connected by NoC as shown in Figure 2. Furthermore, 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 [3] which involves a low average number of hops when compared to mesh topology. 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 which are broken into small pieces called flits (flow control digits).

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 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 are illustrated as nodes in Figure 2.

2.2. Software Model

The software stack used for Kalray MPPA-256 is composed of a hardware abstraction layer, a low-level layer, an OS, and an user application. With respect to scalable computing array of the system, it is possible to map several programming models or runtimes such Linux, a real-time operating system, POSIX API, OpenCL, and OpenMP.

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. 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 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.

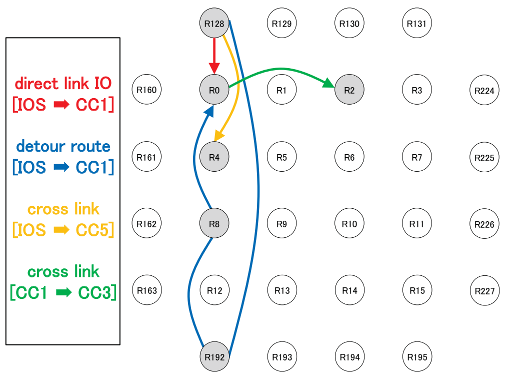


Figure 3. Four D-NoC routes used in the evaluation.

In a low-level library layer, the Kalray system also provides Libnoc and Librouting for handling NoC. 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. Routing on the network is statically conducted with its own policy.

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. Subsequently, we conduct a practical autonomous driving application to examine the practicality of NUMA many cores. The following all evaluations are conducted on real hardware board with eMCOS which is a real-time embedded operating system developed by eSOL (a Japanese supplier for RTOS).

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 3. 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 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, libraries, and an operating system. End-to-end latencies are measured 1,000 times in numerous situations as shown in Figures 4, 5, 6, and boxplots are obtained as depicted in Figures 7 and 8.

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 an IOS and a 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 4, 5, 6, and 7, 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. 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 an 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 8 to facilitate a precise analysis with respect to the interface and memory location. In Figure 8, only the cross-link from the 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 8. 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., 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

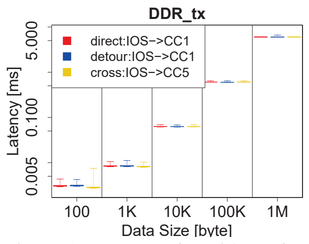


Figure 4. Data transfer with Tx from IO DDR to CC.

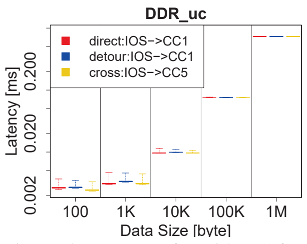


Figure 5. Data transfer with UC from IO DDR to CC.

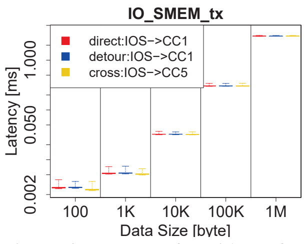


Figure 6. Data transfer with Tx from IO SMEM to CC.

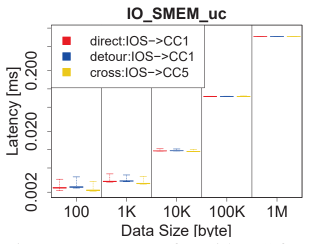


Figure 7. Data transfer with UC from IO SMEM to CC.

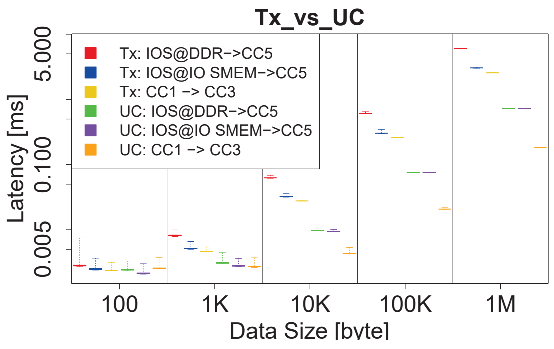
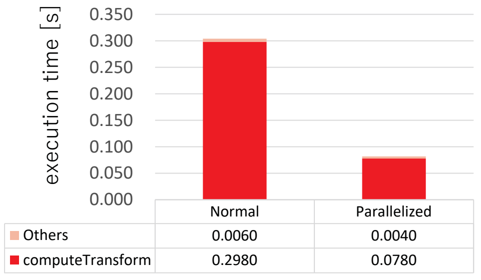


Figure 8. Data transfer with Tx/UC.



	Normal	Parallelized
Others	0.0060	0.0040
computeTransform	0.2980	0.0780

Figure 9. Vehicle self-localization of the partially parallelized autonomous driving application.

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 an IOS and a CC.

3.2. 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], and parallelized part of it. The self-localization adopts the normal-distribution transform matching algorithm [5] implemented in the Point Cloud Library.

The self-localization algorithm is primarily composed of *computeTransform* function which searches for several nearest neighbor points for each scan query and calculates matching transformation. This evaluation parallelized a part of *computeTransform* onto 16 CCs and the remainder of the algorithm was parallelly executed on the IOS with its four cores. To parallelize remainder of *computeTransform* in CCs, 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 and there is room for improvement by the parallelization potential of the MPPA-256.

As shown in Figure 9, the evaluation of 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 achieves the deadline. This parallelized algorithm is executed on simulation and real car experiments in our test course and worked successfully.

4. Conclusions

This work has conducted quantitative evaluations of data transfer methods on NoC and a 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 a real application proofs practicality of NoC-based embedded NUMA many-cores.

In future work, we will port real-time systems such as that in Ref. [6] on CCs and propose the parallelization of memory intensive algorithms such as the nearest neighbor search in Section 3.2.

Acknowledgments

This paper was partly supported by Toyota Motor Company, eSOL, and Kalray.

References

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

[2] 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 ECRTS*, pages 14–24, 2016.

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

[4] 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 DATE*, pages 1–6, 2014.

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

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

[7] 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 RTNS*, pages 235–244, 2016.

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