Exploring Data Transfer with NoC based Many-Core Processors

ABSTRACT

[TBD]

Keywords

Many-core; Multi-core; Network on Chip

1. INTRODUCTION

The next generation computing platform toward multi/many cores is becoming unavoidable to satisfy a increasing demand of computation with reasonable power consumption. In recent years, computation amount of single core processor is constantly reaching its limit and persistence of Moore's law [19] is unclear. Multi/many core architecture is a major trend of interest in the last years. Integrating cores, they realize high-performance and general-purpose computing with low power consumption. This high energy efficiency is the superior feature of multi/many core platforms.

Above increasing demand of computation with reasonable power consumption drives the need for multi/many core architecture in many domains. Real-time embedded systems are one example of adapting multi/many core platforms because they face increasing processing requirements. In this domain, countless opportunities of multi/many cores platforms are discussed [4], [25], [22], [21], [5].

For example, automotive systems contain various applications and some of their demands for high-performance computing. Automotive applications are responsible for many control systems such as the powertrain, the chassis, the steering, driver assistance and user interface. Advanced driver assistance system (ADAS) is becoming more and more complex and is required intelligence such as autonomous driving system. Their complexity and computing requirements are continuously growing. Automotive systems require more computing resources. Meanwhile, it needs to realize extraordinary energy efficiency and reduce the costs. Although modern automotive systems are constructed of a lot of Electronic Control Unit (ECU) managing subsystems, there is a trend from numerous scattered ECUs to hierarchical multi/many core Domain Controllers (DC). Obviously, the hybrid scenario is also available. Combing performance with low power consumption, DCs realize high parallel processing and efficient data transport. In addition, flexible scalability of many-core platform assists development efficiency.

For another example, avionics systems are also responsible for various applications and a need of cost efficiency. With highly integrated trend and increasing demand of processing, multi/many core platforms have opportunities in the avionics domain. Low power consumption of cores leads to lower cooling requirements, which is the critical issue for avionics. Multi/many core platforms reduce not only energy consumption but also the costs and weight. The reduction of weight is the most important result for avionics. Integration with the multi/many core platform reduces the amount of processing board and cooling units. Thus, utilizing integrated platform saves space, weight and the costs.

Considering above background, multi/many core platforms are drawn up and released as commercial off-the-shelf (COTS) multi-core components. They are delivered significant attention in recent years. Kalray's the Multi-Purpose Processing Array (MPPA) 256 [12], [11], [15], Intel's Single-chip Cloud Computer (SCC) [1],[3], and Tilera's Tile64 [2] have clustered many-core architectures, where cores are mapped closely. Their clusters of cores are capable of separate independent applications with desired power envelope of embedded applications. Kalray's MPPA-256 packs 256 general-purpose cores, which is overwhelming the number of other COTS's cores. With high-performance per watt, it targets embedded systems, HPC, image processing, and networking. Intel's Xeon Phi [7] [8] is one of high-performance computing (HPC) accelerators. Xeon Phi is based on x86 architecture and targets use-case of data centers and workstation.

Although the need of multi/many core platforms has emerged, there are several difficulties for adaptation of these platforms to real-time embedded systems [4], [25]. These difficulties are caused by their hardware architecture and strict requirements of embedded systems. One difficulty is that cores share numerous resources (e.g. memory subsystems and I/O devices). Since parallelized complex processes share some resources, contention to these is frequently occurred. Cache coherency is also a critical problem owing to their numerous cores. These difficulties disturb predictable timing behavior and software analysis. Ensuring timing requirement of real-time embedded systems is still an open issue. In multi/many core platforms, the impact of integrating real-

time applications is not completely understood yet. This is a critical issue because embedded systems have requirements for reliable and predictable behavior.

Contributions: In this paper, we clarify the performance characteristics of currently-achievable data transfer methods for many-core computing based on NoC such as MPPA-256 while unveiling several new data transfer methods.

[TBD]

To the best of our knowledge, this is the first evidence of data transfer matters for many-core computing beyond an intuitive expectation, which allows system designers to choose appropriate data transfer methods. depending on the requirement of their latency-sensitive many-core applications. These findings are also applicable for same types of many-core architectures rather than MPPA-256. We believe that the contributions of this paper are useful for low-latency many-core computing.

Organization: The remainder of this paper is organized as follows. First, Section 2 summarizes our considering system model for this paper. We present our hardware model, i.e., Kalray MPPA-256 Bostan, and our system model there. Second, Section 4 illustrates our experimental evaluations. Then, Section 5 presents related work about multi/many core systems. Finally, Section 6 concludes this paper and suggests future work.

2. SYSTEM MODEL

In this section, we present our system model used throughout this paper. We consider the many-core model of Kalray MPPA-256 Bostan. First, the hardware model is introduced in Section 2.1, followed by the 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 (see Figures 1, 2, and 3). The MPPA MANYCORE chip integrates 16 compute clusters and 4 I/O subsystems on NoC. We present the architecture of Kalray MPPA-256 in this section.

2.1.1 I/O Subsystems (IOS)

MPPA-256 has 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 I/O subsystem (IOS) comprises quad core resource managers (RMs) and a network interface.

- **Resource Managers (RMs)**: RM cores are connected to a 16-banked parallel shared memory of 2MB total capacity (IO SMEM), as shown in left side of Figure 1, The four RMs have their own instruction cache 8-way associative of 32 (8×4) KB and share a data cache 8-way associative of 128 KB and external DDR access. Sharing the data cache of 128 KB makes it coherent between the RMs. RM cores operate controllers for the PCIe, Ethernet, Interlaken, and other I/O devices. They are able to operate the local peripherals, including the network interfaces with DMA. We can also make an application run on these RMs.

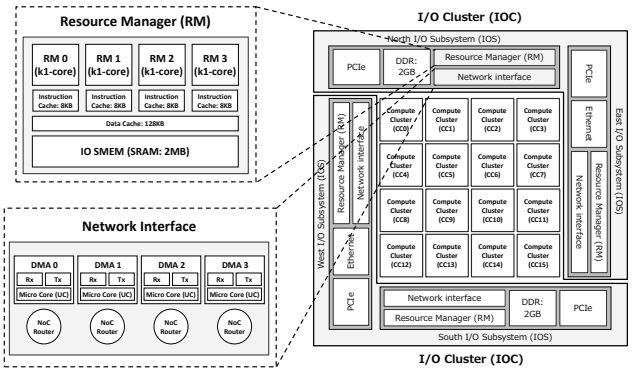

Figure 1: Overview of Kalray MPPA-256 Bostan architecture.

- **A Network Interface**: A network interface contains four DMA engines (DMA0-3) and four NoC routers as shown in left side of Figure 1, the IOS DMA engine manages transfers between the IO SMEM, the IOS DDR, and the IOS peripherals (e.g., PCIe interface and Ethernet controllers). Through NoC Routers, DMA engine transfers data between routers on NoC. The DMA engine has three NoC interfaces: a receive (Rx) interface, transmit (Tx) interface, and micro core (UC). Micro core is a network processor programmable to set threads sending data with Tx interface. UC is able to extract data from memory using a programmed pattern and to send them on the NoC. Once initiated, this is made in an autonomous fashion without using a PE and an RM.

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 (PEs) and an RM**: In a CC, 16 processing elements (PEs) and one RM share 2MB cluster local memory (SMEM) composed of 16 independent memory banks of 16 independent memory banks of $16,384 \times 64$bit. Each bank of SMEM has a capacity of 128 KB. These PEs are mainly used by users for parallel processing. Developers spawn computing threads on PEs. The PEs and an RM in CC are the Kalray-1 cores, which implement a 32-bit 5-issue Very Long Instruction Word (VLIW) architecture with 16W-600MHz (typical) or 24W-800MHz. Each core is fitted with its own instruction and data caches. Each cache is 2-way associative with a capacity of 8 KB. Thus, 17 k1-cores (a PE or the RM) share multi-banked 2MB SMEM.

- **A Debug Support Unit (DSU) and a Network Interface**: In addition to PEs and an RM, bus masters on SMEM are the debug support unit (DSU) and a DMA engine in a network interface. A DMA engine and a NoC router are laid out in a network interface. Similar to IOS, the CC DMA engine has also three interfaces: an Rx, a Tx, and a UC. It is instantiated in every cluster and connected to the SMEM.

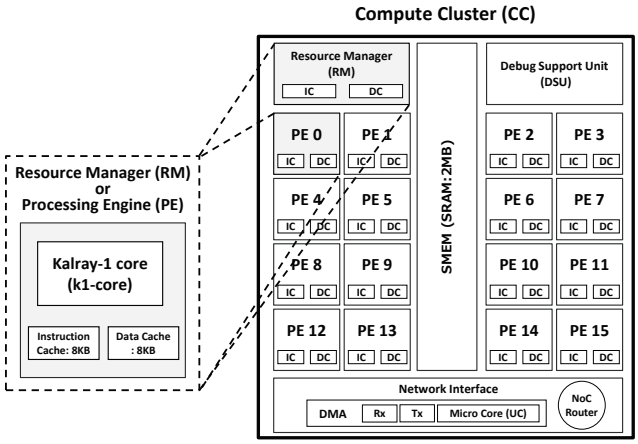

Figure 2: Compute Cluster (CC) architecture.

2.1.3 *Network-on-Chip (NoC)*

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

- **Bus Network**: Bus network connects nodes (CCs and IOSs) with torus topology [9] which has twice bands and takes a low average number of hops compared to mesh topology [27], [26]. The network is actually composed two parallel NoC with bi-directional links (red lines in Figure 3): the data NoC (D-NoC), which is optimized for bulk data transfers, and the control NoC (C-NoC), which is optimized for small messages at low latency. Functionally, NoC is a packet switched network. Data are packaged in variable length packets which circulate between routers in a wormhole manner: packets are broken into small pieces called flits (flow control digits). NoC traffic is segmented into packets: each packet has 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 two own routers: a D-NoC router and a C-NoC router Each RM on NoC node (CC or IOS) is associated with these two NoC routers. DMA engines in a network interface on CC/IOS send and receive flits through these D-NoC routers with the Rx interface, the Tx interface, and UC. A mailbox component which is the virtual interface for the C-NoC enables one-to-one, N-to-one, or one-to-N low latency synchronizations. 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 simplicity, we illustrate D-NoC/C-NoC routers with one NoC router. In both D-NoC and C-NoC, each network node (CC or IOS) has 5-link NoC routers: four duplexed links for north/east/west and south neighbours, one duplexed link for local address space attached to the NoC Router. NoC routers have FIFOs queuing flits for each direction. The data links are four bytes wide in each direction and operate at the CPU clock rate of 600MHz or 800MHz; therefore, each tile can transmit/receive a total of 2.4GB/s or 3.2GB/s spread across the four directions (i.e., north, south, east, and west).

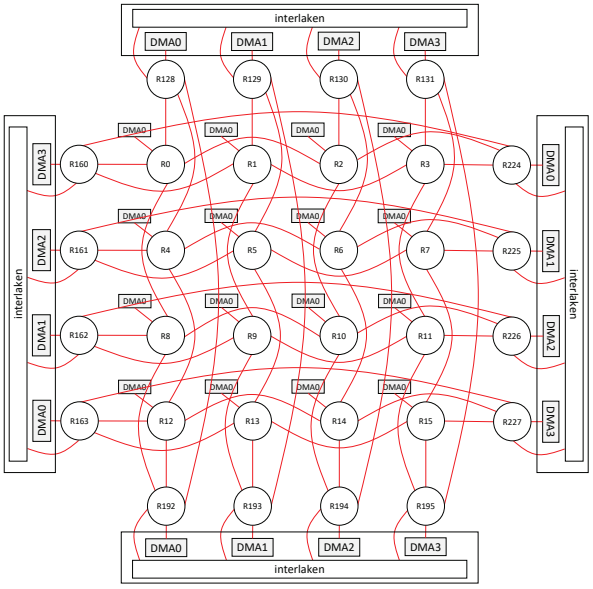

Figure 3: Network-on-Chip (NoC) connections (both D-NoC and C-NoC).

2.2 Software Model

A software stack for Kalray MPPA-245 used in this paper is shown in Figure 4. Kalray's MPPA system is an extensible and scalable array of computing cores and memory. On such a system, we can map several programming models or runtimes, e.g., Linux, real-time Operating System, POSIX API, OpenCL, OpenMP. We describe each layer in detail.

In hardware abstraction layer, an abstraction package abstracts hardware of a CC, IOS, and NoC. This abstraction package serves as a system that doesn't provide any services. The hardware abstraction is responsible for partitioning the hardware resources and controlling access to these resources from the user-space operating system libraries. Moreover, the abstraction package is able to retrieve the resources allocated to a partition at any time. It is able to set-up and control inter-partition communications as virtual machine abstraction layer. The hardware abstraction runs on the dedicated RM core. All the services are commonly provided by a rich operating system (virtual memory, interrupts, scheduler, etc...) must be provided by user-space libraries. Consequently, each runtime or operating system implements its own required services optimized to its needs. This is because each programming model or runtime has different requirements. Minimal kernel avoids the waste of resources and mismatched needs.

In low-level Library layer, Kalray also provides libraries for handling NoC. NoC features such as routing, quality of service are to be set by the programmer. The Libnoc allows direct access to memory mapped registers for their configurations and use. It is designed to cause a minimum amount of CPU overhead. It also serves as a minimal abstraction for

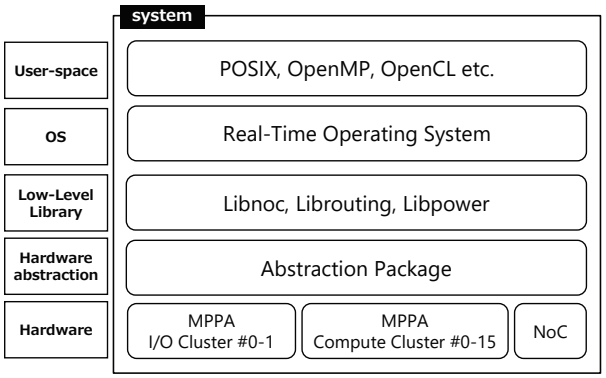

Figure 4: Kalray MPPA-256's software stack.

resource allocation. The Librouting offers the minimal set of functions to use for routing data between any clusters of the MPPA, both with unicast (one target) mode or multicast (multiple targets) mode. Routing on the torus network shown in Figure 3 is statically conducted with its own policy. The Libpower enables spawning and waiting for the end of execution of a remote cluster.

In OS layer, various operating systems support the abstraction package. We introduce the following Real-Time Operating System (RTOS):

- **RTEMS**: RTEMS, the Real-Time Executive for Multiprocessor Systems, is a full featured RTOS prepared for embedded platforms. It supports several APIs and standards, and most notably the POSIX API. The system provides a rich set of features, and an RTEMS application is most of the time just a regular C or C++ program using the POSIX API. We are able to build RTEMS on IOC.

- **NodeOS**: On CC, we can build the MPPAcluster operating system a runtime called NodeOS. This OS addresses the need for a multicore OS conforming as much as possible to the standard POSIX API. The NodeOS enables user code using POSIX API to run on PEs on CC. First, NodeOS runtime starts on PE0 before calling the user main function. Then, we can call pthread on other PEs.

- **eMCOS**: On both CC and IOS, eMCOS provides minimal programming interfaces and libraries. eMCOS is a real-time embedded operating system developed by eSOL, a Japanese supplier for RTOS. eMCOS is the world's first commercially available many-core RTOS for use in embedded systems. This 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 not only IOS but also CC.

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

3. DATA TRANSFER FRAMEWORK

In this section, we explain data transfer methods in MPPA-256. For scalability, MPPA-256 accepts clustered architectures where each cluster contains own memory. 16 cores are packed as a cluster and they share 2 MB memory (SMEM) as shown in 2. Avoiding frequent memory contention by numerous cores, this helps increment of the number of cores. However, this architectures constraints memory address which cores are able to access directly. To communicate with cores outside the cluster, we have to transfer data between clusters through D-NoC with network interfaces.

In receiving side, there is Rx interface to receive data with DMA. We have to allocate a D-NoC Rx resource and configure it to wait receiving data. One DMA in a network interface contains 256 D-NoC Rx resources. In sending side, we have two interfaces for users to send data between clusters. One is Tx interface and the other is UC interface as explained in Sections 2.1.1 and 2.1.2. UC is a network processor programmable to set threads sending data in DMA. It executes programmed pattern and sends data through D-NoC without a PE and an RM. UC interface provides faster data transfer than Tx interface. However, One DMA in a network interface contains only 8 D-NoC UC resources. Both interfaces use DMA engine to access memory and copy data. Regardless of using UC interface or not, we have to allocate a D-NoC Tx resource and configure it to send data. If we use UC interface, we additionally allocate and configure a D-NoC UC resources.

4. EVALUATIONS

In this section, we have conducted two kinds of evaluations. One is D-NoC data transfer evaluation. We explore latency characteristics of interfaces and memory type. The other is matrix calculation evaluation. This test shows MPPA-256's parallelization potential and characteristics of memory access when dealing with large data.

4.1 D-NoC Data Transfer

In this evaluation, we clarify end-to-end latency considering the relation among interfaces (Tx or UC) ,routing on NoC, and memory type (DDR or SMEM). To achieve above purpose, we prepare four routes as shown in Figure 5. These routes on D-NoC map (Figure 3) contains various connections between routers; direct link, cross link, and flying link. In case of routes from the IOS routers to the CC routers, transmitted data is allocated in DDR or IO SMEM. CC has only SMEM as shown in Figure 2. We directly use low-level library to transfer data with D-NoC. Transferred data are 100 B, 1 KB, 10 KB, 100 KB, and 1 MB. These buffers are sequentially allocated in DDR or SRAM (IO SMEM or CC SMEM). Since the capacity of CC SMEM is 2 MB, we assume that appropriate communication buffer size is 1 MB. On our assumtion, the other memory area is own by application, library, and operating system. In many situations, we have measured end-to-end latencies 1,000 times and drow boxplots in Figures 6, 7, 8, and 9, 10 and 11. Following evaluations is conducted on eMCOS.

Data transfer latencies between IOS and CC is not influenced by routing. We prepare two interfaces (Tx and UC), three routes(direct link, cross link, and detour route), and two memory location where transferred data is allocated. As shown in Figures 6, 7, 8, and 9, end-to-end latencies scales

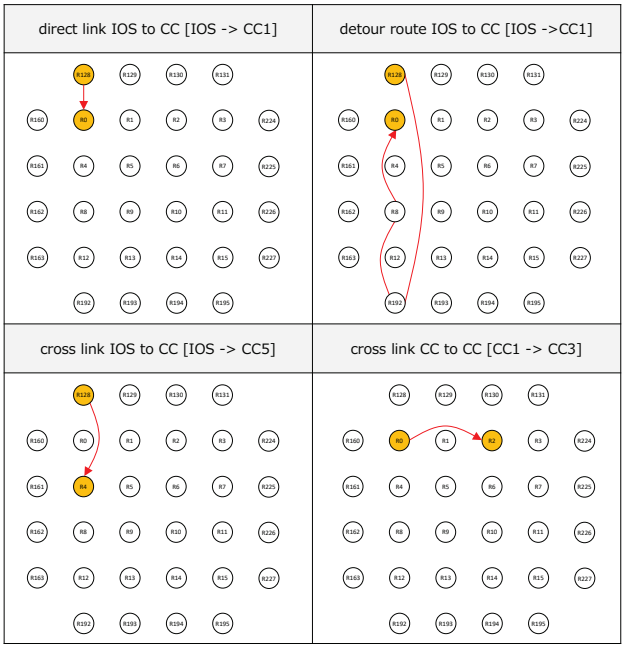

Figure 5: Four D-NoC routes used for evaluation.

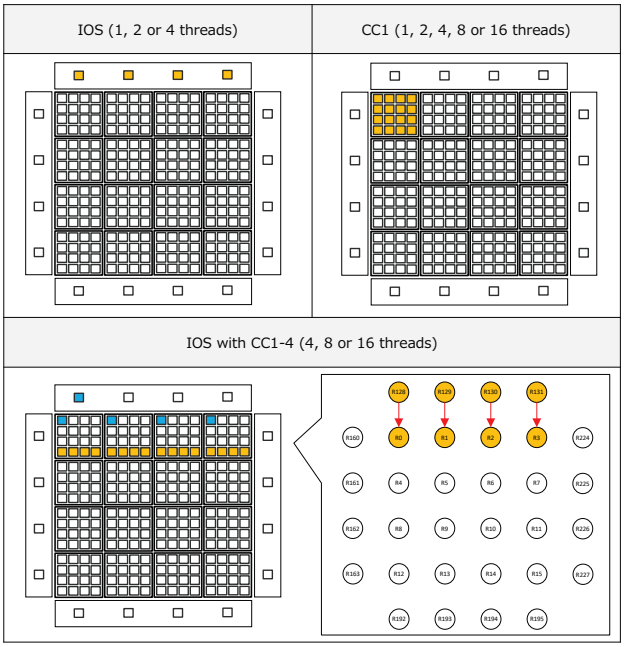

Figure 12: Situations of matrix calculation.

linearly with data size and three routes do not produce differences in data transfer latency. This result is important in torus topology NoC because the number of its minimum step is larger than mesh topology's one. We can observe that queuing in NoC Routers and hardware distance on NoC is not dominant factor for latency. Transmitting and receiving transaction take much longer time than other transaction. In addition, we briefly recognize that UC is much faster than Tx. For precise analysis for interface and memory location, we arrage data in Figures 10 and 11. In these figures, we accept only cross link from IOS to CC5 because routes do not influence latency. For intuitive recognition, we arrange two kinds of figures; logarithmic axis one and linear axis one.

In Tx interface, DDR causes a large increase in latency. DDR takes twice time as long as IO SMEM as shown in Figure 11. This is caused by memory access speed characteristics of DRAM and SRAM. In case of Tx interface, an RM (k1-core) on IOS has to operate DMA in network interface of IOS. It is thought that the fact that the core is involved in processing is causing this result. Data transfer latency between CCs is a little faster than that between IOS and CC. This result indicates that MPPA-256 is optimized for communication between CCs.

In UC interface, whether transferred buffer is allocated on DDR or SMEM has not much effect on latency. Same latency characteristics is observed in Figures 10 and 11. In case of UC interface, an RM (k1-core) on IOS does not involve DMA transaction. A micro core in network interface executes a programmed thread sending data. This evaluation result tells that there is no need to be conscious of the slow access speed of DDR in case of UC. As well as Tx interface, Data transfer latency between CCs is faster than that between IOS and CC.

4.2 Matrix Calculation

In this evaluation, we clarify matrix calculation time and parallelization potential of MPPA-256. We have conducted matrix calculation in IOS and CC. As shown in Figure 12, three computing situations is considered. One is computing in IOS where four cores are available. To analyze memory access characteristics, we allocate matrix buffer in IO DDR and SMEM. Another is computing in CC where 16 cores are available. The other is offload-computing using an IOS and four CCs. Parallelized processing is executed with four CCs cores and there SMEM. Some cores in IOS and CC manage the parallelized transaction. This method is able to handle large data which one cluster cannot deal because buffer capacity is not limited to 2MB of SMEM. Parallelized processing and the total capacity of SMEM is superior to computation in IOS or CC. In IOS, application can handle large capacity data only in DDR. However, with this method, we can deal with large capacity data in SMEM by distributed memories. For faster data transfer, a part of matrix buffer is parallelly transmitted as shown in Figure 12. To avoid cache coherency trouble, IOS and CC cores must access matrix buffers without cache.

We analyze matrix calculation time with parallelization and memory allocation. In addition, we analyze the influence of cache because cache coherency is an important issue in many-core system. There are many cases that applications must access specific memory space without cache. Since maximum buffer size is 1 MB in our assumption, we prepare three matrices buffers and each size is 314 KB. The matrix A and the matrix B are multiplied, and the result is stored in the matrix C. We set the total of the three matrices to be approximately 1 MB.

First, matrix calculation time with cache in IOS and CC is shown in Figure 13. Thanks to cache, there is almost no difference between IO DDR, IO SMEM, and CC SMEM. 128 KB data cache in IOS works well and compensates for the delay of DDR. Additionally, we can observe that calculation time scales linearly with the number of threads. This is ideal

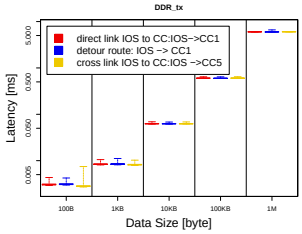

Figure 6: Data transfer with Tx from IO DDR to CC.

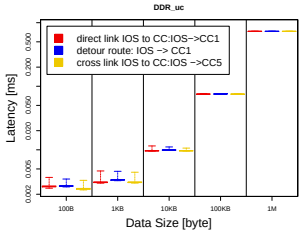

Figure 7: Data transfer with UC from IO DDR to CC.

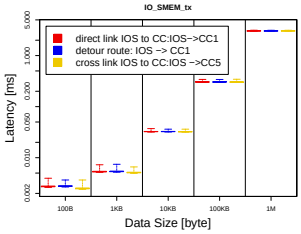

Figure 8: Data transfer with Tx from IO SMEM to CC.

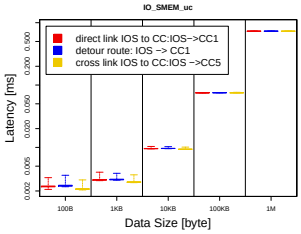

Figure 9: Data transfer with UC from IO SMEM to CC.

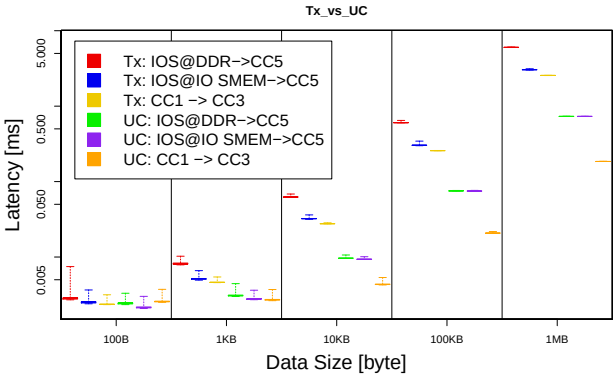

Figure 10: Data transfer with Tx/UC (logarithmic axis).

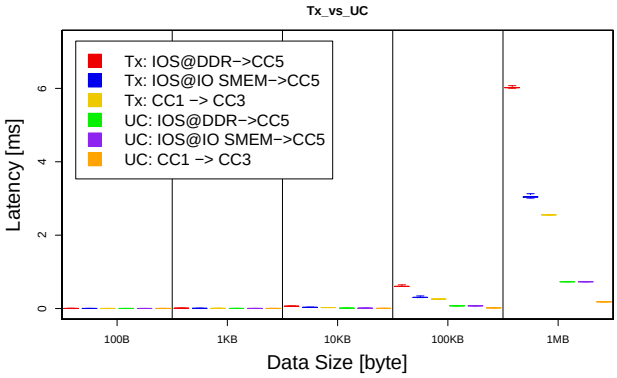

Figure 11: Data transfer with Tx/UC (linear axis).

behavior with parallelization.

Second, matrix calculation time without cache in IOS and CC is shown in Figure 14. Due to the lack of cache, DDR has increased about fourfold and a big difference with SMEM has arisen. Another notable result is that calculation in CC SMEM is faster than that in IO SMEM. This characteristic hides in calculation with cache. Since computing cores are same k1-cores in IOS and CC, it is thought that the characteristics and physical arrangement of SMEM are affecting. This is a interesting result that there is a big difference that can not be ignored. We can also observe that calculation time scales linearly with the number of threads. The other notable result is that calculation without cache in CC SMEM is faster than that with cache. Although this result is contrary to intuition, two factors are conceivable. A small data cache (8 KB) in CC does not adequately work and direct assembly instruction for uncached access optimizes memory access.

Finally, matrix calculation with offload-computing in IOS and CCs is is shown in Figure 15. In this case, we suppose that large matrices calculation that total capacity is over 1 MB. Because of this assumption, the offloading result is compared with IO DDR (cashed). To offload a part of calculation onto four CCs and aggregate calculation result, overhead transaction (inverse matB and input matC) occurred. They produce constant overhead regardless of the number of threads as shown in Figure 15. However, offloading result is faster than that of IO DDR (cashed). This result proves several important facts. One is that D-NoC data transfer produce little overhead latency. The other is that DMA memory access to DDR is much faster than an RM (k1-core)'s memory access. In the offloading case, a DMA accesses matrix buffers on DDR and transfer them from IO DDR to each CC SMEM. Then, PEs in CC access matrix buffers for calculation without cache. Since the overhead of data transfer and DMA memory access is small, parallel data transmission and distributed memory are practical in MPPA-256. The impact of offloading becomes larger when the matrix size is large as shown in Figure 16. Since only a part of matrix is allocated in CC, we can handle larger matrices buffers. We prepare 640 KB matrices and evaluate matrix calculation with offload-computing. Compared to result of 314 KB matrices in Figure 15, offloading result is much faster than IO DDR result in Figure 15.

5. RELATED WORK

In this section, we compare many-core platforms to additional platforms and discuss previous work of multi/many cores.

In recent years, computation performance of a single core processor is constantly coming to its limit. Pollack has said inefficiency of a single core in [23] and Moore's law [19] has become unstable. To satisfy increasing a demand of computation, CPU is not sufficient. Many other platforms including many-core are developed and researched nowadays.

Table 1 summarizes features of many-core platforms with that of other platforms. For instance, GPU is a powerful device to enhance computing performance. In specific area (e.g., image processing, learning), it has great potential. However, it is mainly used for a specific purpose and its reliability is not suitable for real-time systems. It is difficult to use GPU for a global purpose and a guarantee of reliability due to GPU architecture. For a global purpose and multiple instructions, many-core is much superior to GPU. In addition, it is commonly known that many-core has reasonable power consumption. GPU consumes a lot of power and generates much heat. This is a critical problem for embedded systems. DSP and FPGA are also high-performance devices compared to CPU. They are efficient in a point of power consumption. DSP is often used for real-time system and FPGA guarantees reliability and efficient processing. They

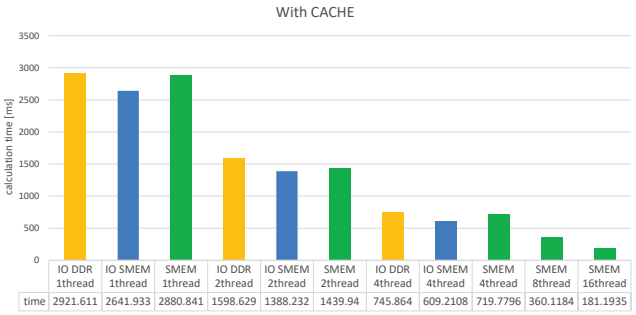

Figure 13: Matrix calculation in IOS and CC with cache.

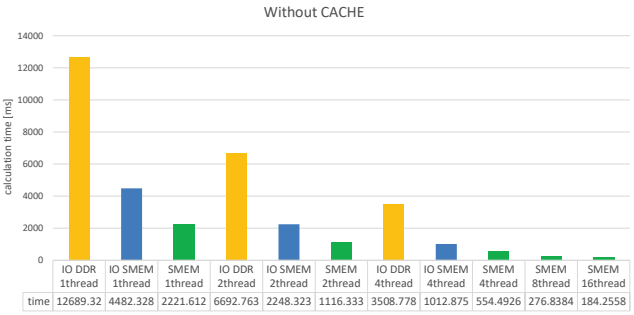

Figure 14: Matrix calculation in IOS and CC without cache.

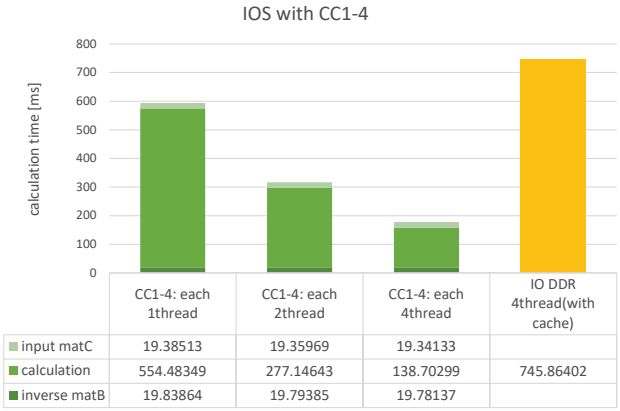

Figure 15: Matrix calculation with offload-computing (314 KB matrix x 3).

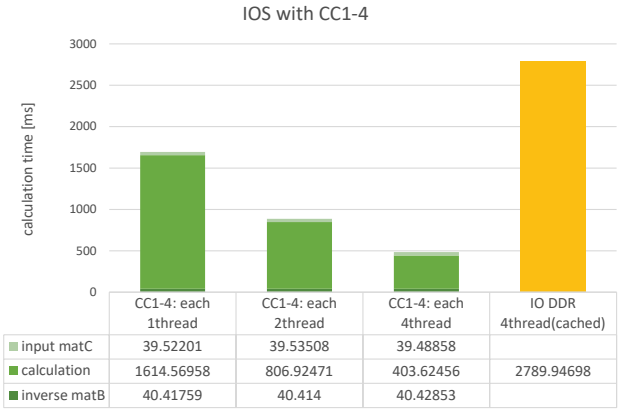

Figure 16: Matrix calculation with offload-computing (640 KB matrix x 3).

are suitable for time-critical computing [10]. However, DSP cannot be used for global purpose and programming FPGA is difficult for software developers. Their software model is much different from CPU and not a substitute for CPU. Many-core platforms have a potential to take the place of single/multi core CPU. Ease of programming and scalability with high acceleration is highlighted.

Based on the above background, real-time applications on many-core platforms has received significant attention in recent years. Many commercial off-the-shelf (COTS) multi-core components are developed and released by several vendors. (e.g., Kalray's the Multi-Purpose Pro-cessing Array (MPPA) 256 [12], [11], [15], Intel's Single-chip Cloud Computer (SCC) [1],[3], Tilera's Tile64 [2], and Intel's Xeon Phi [7] [8]) The Kalray MPPA-256 is designed for real-time embedded applications and the target of this paper. Kalray S.A [12], [11], [15] has presented clustered many-core architectures on NoC. (Precise hardware model is described in Section 2.1.) It is often accepted for a target of many-core platforms and various previous work has considered this model [22], [4], [6], [21].

[TBD]

6. CONCLUSION

[TBD]

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	✓	✓	✓	✓	✓	△	✓