PAPER Special Section on Reconfigurable Systems

A Prototype System for Many-Core Architecture SMYLEref with FPGA Evaluation Boards

Son-Truong NGUYEN[†], ***Nonmember***, **Masaaki KONDO**[†a)], ***Member***, **Tomoya HIRAO**[††*], ***Nonmember***, ***and*** **Koji INOUE**[††], ***Member***

SUMMARY Nowadays, the trend of developing micro-processor with hundreds of cores brings a promising prospect for embedded systems. Realizing a high performance and low power many-core processor is becoming a primary technical challenge. Generally, three major issues required to be resolved includes: 1) realizing efficient massively parallel processing, 2) reducing dynamic power consumption, and 3) improving software productivity. To deal with these issues, we propose a solution to use many low-performance but small and very low-power cores to obtain very high performance, and develop a referential many-core architecture and a program development environment. This paper introduces a many-core architecture named *SMYLEref* and its prototype system with off-the-shelf FPGA evaluation boards. The initial evaluation results of several SPLASH2 benchmark programs conducted on our developed 128-core platform are also presented and discussed in this paper.

key words: *many-core processor, evaluation platform, prototyping, FPGA*

1. Introduction

In recent years, the micro-processor organized with tens of processor cores (also simply called *cores*) on a single chip has been widely used in many fields. In the near future, a prospect of many-core architecture which consists of hundreds of cores is expected. Therefore, developing many-core architectures and compiler techniques is becoming a major technical challenge to realize a high performance and low power many-core processor as well as to develop applications for embedded systems on it.

We are currently developing a high-performance and low-power many-core processor architecture and compiler technology for embedded systems as a part of the NEDO's project. In this research, we propose a solution to use many low-performance but small and very low-power cores to obtain very high performance with high parallel processing efficiency. Based on research and development with these concepts, we aim to realize a low-power cost effective embedded processor as an alternative for the current system-on-a-chip designs. Towards this goal, we focus on the following three major challenges required to be addressed.

1. Realizing efficient massively parallel processing, especially focusing on embedded system applications.
2. Reducing dynamic power consumption.
3. Improving software productivity.

To deal with these issues, we have developed a referential many-core architecture and a program development environment. The many-core processor is used as a platform for the code accelerators. The key point of our research is that, the concept of *Virtual Accelerator on Many-core (VAM)*, in which the many-core processor consisting of many small cores, is utilized as a hardware platform for realizing multiple virtual accelerators. In this platform, a compiler or a user program first decides the configuration of the virtual accelerator (the number of cores, L2 cache configuration, etc.) according to the characteristic of a target code, and then performs code generation suited for that configuration. Since the decision of the hardware architecture is partly granted to the software, higher parallel processing efficiency can be achieved.

In order to increase the scalability of parallel performance for multi-thread or multi-programmed applications, multiple VAMs are enabled to be mapped simultaneously to groups of cores as shown in Fig. 1. For the power efficiency, each core is assumed to be operated with ultra-low supply voltage, which will be realized by a recent improvement of the near-threshold voltage computing technology. In near-threshold voltage computing, the increased performance variation is a big problem required to be resolved. Plenty of hardware resources in many-cores can help reduce the impact of the variation. For the software, we develop an API and a programming environment for parallel programming and optimization for VAM.

As a platform for research and development of a many-

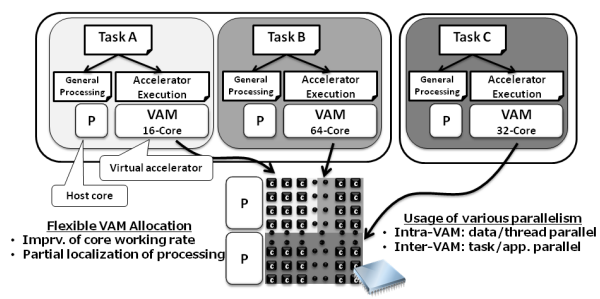

Fig. 1 Concept of Virtual Accelerator on Many-core (VAM).

Manuscript received November 9, 2012.
Manuscript revised February 27, 2013.

[†]The authors are with the Graduate School of Information Systems, The University of Electro-Communications, Chofu-shi, 182–8585 Japan.

[††]The authors are with the Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka-shi, 819–0395 Japan.

[*]Presently, with the Fixstars Corporation, Japan.

a) E-mail: kondo@is.uec.ac.jp

DOI: 10.1587/transinf.E96.D.1645

core processor and VAM, we are developing a many-core architecture called *SMYLEref* and its prototype system with off-the-shelf FPGA evaluation boards. We use this prototype system for a software development environment and evaluating proof of concept of the VAM. Usually, software simulators are used for architecture researches. Though the software simulators are flexible and easy to use, the scalability is not sufficient. Simulation for a many-core processor with hundreds of cores takes long time and is not realistic. On the other hand, the emulation of many-core processor with FPGAs is fairly practical because of its evaluation speed and scalability. In this paper, we introduce the architecture of SMYLEref and its prototype system developed on FPGA boards. In addition, several initial experiments with the prototype system are also presented.

The remainder of this paper is organized as follows. Section 2 describes the detail of SMYLEref architecture. The construction of evaluation platform developed on FPGA is presented in Sect. 3. Several implementation and evaluation results, and related works are discussed in Sect. 4, and Sect. 5. Finally, Sect. 6 concludes the paper.

2. Many-core Architecture *SMYLEref*

The block diagram of SMYLEref is shown in Fig. 2. In the SMYLEref architecture, multiple blocks called *clusters* are arranged in a two-dimensional array and connected by a two-dimensional mesh on-chip network (NoC). Each cluster is composed of a number of scalar processor cores (for example, 8 cores as in Fig. 2) which are combined by a bus connection. Each core has a dedicated L1 instruction cache (IL1) and an L1 data cache (DL1). An L2 cache is provided for each cluster and is shared by all the cores in a chip.

2.1 Processor Core

In SMYLEref, we use a processor core named *Geyser* [2], which is developed in a JST CREST research project called "Innovative Power Control for Ultra Low-Power and High-Performance System LSIs" [1] for creating clusters. Geyser is a simple processor core based on the MIPS R3000 architecture. The design of Geyser core is verified in the real LSI chip and the Linux operating system can be launched

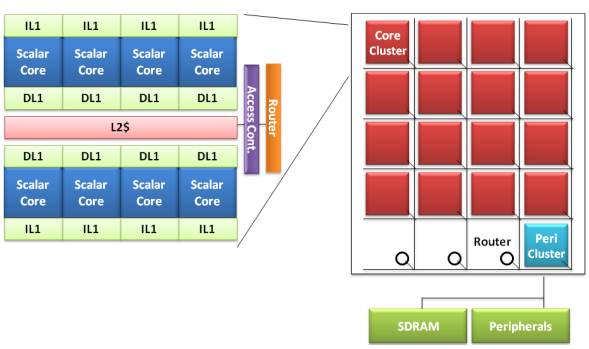

Fig. 2 Architecture of SMYLEref.

on it. Therefore, it is adopted as a base processor core for our platform. Note that Geyser is originally developed for fine-grain power gating research and has capability of run-time power-gating [4]. However, this function is not used in SMYLEref.

Geyser core consists of a simple 5-stage pipeline, 8 KB (2-way set associative, 64 B line size) L1 instruction and L1 data caches, and a 16-entry TLB (Translation Look-aside Buffer) which can be controlled by instructions. The data cache adopts the write-back policy.

2.2 Structure of Cluster and NoC

There are two types of clusters in SMYLEref: one is the *core cluster* with Geyser processor cores and an L2 cache, and the other is *peripheral cluster* which plays a role of an external interface of the chip. In the core cluster, multiple Geyser cores are connected through a cluster bus. The L2 cache and a router for the network-on-chip are also connected to this bus. A peripheral cluster consists of an SDRAM controller, several I/O controllers, and a router. The access to the main memory and I/O devices is ensured by packet-based data transfer which is performed between the core cluster and the peripheral cluster through an on-chip network.

Core clusters and peripheral clusters are connected by a two-dimensional mesh network-on-chip which is formed out of virtual channel routers. The packet-switching mechanism is utilized for data transfer in the network. By hierarchically organizing processor cores, that is, multiple cores with L2 cache in a cluster and using the network for connecting clusters, we intend to confine the communication within a cluster as much as possible and avoid the performance degradation caused by traffic congestion in the network. Although multiple peripheral clusters can be configured for SMYLEref architecture, in our prototype system, only one peripheral cluster is used for simplicity of implementation. If multiple peripheral clusters are implemented, the destination cluster should be identified in the source core cluster according to the accessed physical memory addresses. When the number of peripheral clusters used for the system is smaller than the number of nodes necessary to make a row (or a column) of a mesh network (for example, four nodes in Fig. 2), a special node which has only a router should be filled in for the deficient cluster location in order to form the regular mesh connections. This ensures the integrity of XY routing.

In SMYLEref, the two-dimensional mesh network is adopted for connection between clusters because of its scalability, effective communication locality, as well as extendability. The *XY routing* algorithm is adopted for guaranteeing deadlock-free routing and the *iSLIP* scheduling algorithm is employed for the switch allocation due to its high performance, flexibility and no starvation property [5]. For data transfer, the virtual-channel flow control [8] is used. The packets are divided into multiple flits (flow control digits) where the two-flit header (i.e. the header includes two flits) contains all the necessary information for routing and

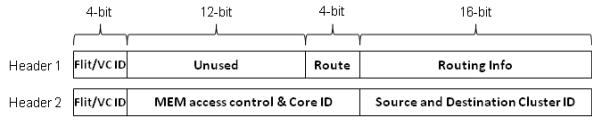

Fig. 3 Format of the two-flit header.

memory accesses including a route for the next hop (the channel selection signal for the succeeding router), cluster and core IDs, and memory access controls (read/write, non-burst/burst, etc.). The other flits of the packet carry payload data. The format of two-flit header is illustrated in Fig. 3 where *Flit/VC ID* indicates the flit and virtual channel IDs. As shown in the figure, each cluster is distinguished by a unique cluster ID, and each core within a cluster can be identified by *core ID*. To reduce network latency, we utilize the look-ahead technique [8], in which the route for the next hop is pre-computed one step ahead. This information is included in the header (*route*). In the next router, the packet can use it to directly proceed to the virtual channel allocation stage for determining its output channel without waiting for a routing computation. Because the pre-computation of the route can be performed simultaneously with other pipeline stages, it does not affect the critical path length of a router. By performing all pipeline stages of the router in parallel, the header is updated at the same time the packet is transferred to the next hop in only one clock cycle. Therefore, no additional pipeline stage is needed for the header update.

To form the on-chip network, we utilize the On-the-fly router [6], which is an enhanced design of the conventional virtual channel (VC) router architecture. This router can perform high performance data transfer with low hardware cost which may contribute to energy-efficiency. Basically, an on-chip router forwards incoming packets through a critical path of four pipeline stages including the routing computation (RC), virtual channel allocation (VA), switch allocation (SA) and switch traversal (ST). The delay of the critical path at each node is one of the causes leading to the network latency. Several proposed approaches allow to speculatively perform VA and SA in parallel to cut down the critical path [7], [8]. However, it leads to the degradation of network throughput caused by the speculation. Different from that, in the On-the-fly router, instead of speculatively performing the VA in parallel with the SA, the router only performs the VA for the packets which have won the switch arbitration. A VC is allocated for a packet during the time the packet is traversing the crossbar switch. With this constraint, the dependency between the VA and ST is removed, and these stages can be concurrently performed without the speculation. As a result, there is no impact of speculation on throughput even in the heavy network loads, and the efficiency of channel utilization is significantly enhanced.

The block diagram of the router used in SMYLEref is shown in Fig. 4. The router has five bi-directional ports named as North (N), South (S), East (E), West (W), and Local (L) for communicating with the neighboring routers and its cluster bus, with data bit-width of 32-bit and two virtual channels in each port.

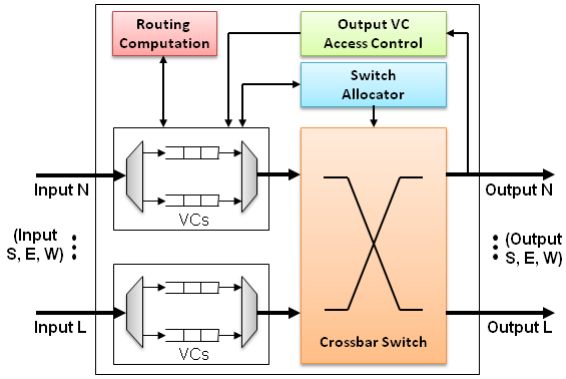

Fig. 4 Architecture of Router.

3. Evaluation Platform on FPGA

In development and verification of architectures and softwares in many-core era, the evaluation platform plays a very important role. Functional level or cycle level software simulators [9]–[11] have been widely used for architecture research and exploration environment so far. Most of the software simulators simulate the target programs sequentially, even when the multi-threaded and/or concurrent programs are simulated on a host machine with multiple processors. When simulating many-core processors with tens to hundreds of cores, a quite long simulation time is needed even for evaluating only a part of the target program. Moreover, if the OS code is also taken into account for evaluation, the simulation time will be much longer.

One of the alternatives to the software simulators is the real LSI implementation. This approach enables to reduce the evaluation time significantly and ensures the accuracy of evaluation results. However, it is not flexible and requires very high cost of development. Also, it is not easy to extend for various evaluation models of different many-core architectures.

Another solution with the use of FPGA is becoming a reasonable alternative because of its scalability, accuracy and flexibility. This solution allows to improve the speed of evaluation considerably as compared to that of software simulators, while still maintaining an acceptable cost of development. By implementing multiple actual cores operating in parallel on FPGA, it is possible not only to reduce the necessary time for evaluation, but also to allow the real execution of an operating system for verification. In addition, it is not difficult to develop additional hardware modules at register transfer level to extend the evaluation environment. Multiple FPGA devices can also be easily combined to create a larger system. The comparison among evaluation platforms of software simulator, LSI implementation, and FPGA prototyping in terms of scalability, accuracy, flexibility, development cost and evaluation speed is summarized in Table 1.

Due to mentioned-above remarkable advantages, we chose the solution of developing evaluation platform using FPGA for evaluating many-core architecture.

Table 1 Comparison among evaluation platforms of software simulator, LSI implementation and FPGA prototyping.

	Scalability	Accuracy	Flexibility	Development Cost	Evaluation Speed
Software Simulator	Low	Medium	Very High	Low	Low
LSI Implementation	Medium	Very High	Low	Very High	Very High
FPGA Prototyping	Very High	Very High	High	High	High

3.1 Hardware and Developing Environment

The Xilinx ML605 evaluation board which has a Virtex-6 FPGA device, an SDRAM, and several I/O interfaces such as UART, SysACE, etc. is used as a basic component for our platform. The specifications of the ML605 evaluation board and the Virtex-6 device are summarized in Table 2. We use Verilog-HDL for the circuit design and the Xilinx ISE for the Verilog-HDL simulation and implementation to the FPGA.

3.2 Implementation of Core and Peripheral Cluster

The overview of the core cluster is depicted in Fig. 5. Each core cluster is formed from several Geyser cores (up to eight cores) and an On-the-fly router as described in Sect. 2.2. In addition, the L2 cache is also located on the core cluster. The L1 data/instruction caches, the L2 caches, and router buffers are implemented by using dedicated BlockRAMs (BRAMs) of FPGA. The cluster bus is 32-bit width which is the same as bit-width of the Geyser core bus. The packet width in the router used for communicating with its neighbors is 38-bit, in which 32-bit is dedicated for data and another 2-bit is used for the flit identification of the packet and the remainder is needed for buffer credits of the router. The memory accesses caused by L1 cache misses are controlled by a module named *MEM Access Controller*. If the address of a memory access is of the cacheable property, the access goes to the L2 cache and if the L2 cache miss happens, then the access goes to the main memory via the router.

Originally, the Geyser processor core does not have any floating-point arithmetic co-processor. To evaluate the floating-point applications, FPU (Floating Point number Unit) co-processor is also developed for our evaluation platform. Note that, with the use of FPUs, the number of cores in each core cluster should be scaled down. We also implement a coherence control mechanism within a core cluster using a simple snoop-based protocol.

The peripheral cluster is composed of several controllers of peripherals such as the SDRAM, the UART, and the SysACE which are combined by Xilinx PLB (Processor Local Bus). We use IP cores provided by Xilinx for those controllers. Similar to the core cluster, a router is also associated with peripheral cluster, and linked to the PLB. By this way, the peripherals will be connected to core clusters via the on-chip network. As an enhancement, our prototype system enables to set the additional memory access latency arbitrarily, which contribute to the evaluations on the impact of memory access latency.

Table 2 Specification of ML605 board and Virtex-6 chip.

ML605 evaluation board	
FPGA device	Virtex-6 XC6VLX240T
SDRAM	DDR3 SO-DIMM
I/O ports	GTX, UART, USB, DVI, SysACE, SMA, etc.
Clocks	200 MHz & 66 MHz oscillators
Virtex-6 (XC6VLX240T)	
Technology	65 nm CMOS, 1.0 V
Logic Cells	241,152
CLB Slices	37,680
Block RAM	14,975 Kbit
User I/Os	720

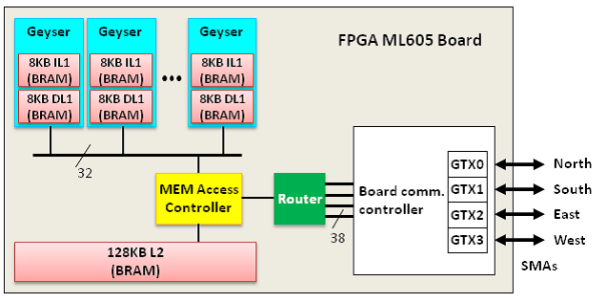

Fig. 5 Configuration of a core cluster.

3.3 Inter-Cluster Communications

Since the hardware resource of the FPGA device (Virtex-6 XC6VLX240T) used for our evaluation platform is exhausted with only about eight Geyser cores, a many-core processor with tens or hundreds of cores cannot be evaluated on one ML605 board. Hence, our evaluation platform is constructed by a combination of multiple ML605 boards connected in a network, in which each core cluster is implemented in a board and the communication between routers is performed by a communication interface between boards. Note that the peripheral cluster is a special case. Since the hardware resource usage of the peripheral cluster is very small, it is configured in the the same board in accordance with one of the core clusters.

We use a high-speed serial communication interface called *rocket I/O* for communication between boards. The rocket I/O interface uses GTX transceivers for its data conversion and transmission. A rocket I/O communication module is developed as a wrapper for the router module. When a router transfers some packets to a neighboring cluster, the data is transmitted between boards via the rocket I/O interface. This module uses a high-speed serial communication protocol called *aurora* provided as an IP core in Xilinx ISE. The SMA (SubMiniature version A) standard interface is used as a physical connection for serial communication

between boards.

Although ML605 evaluation board has only two links (bi-directional) of SMA connection, it is possible to increase up to eight links for one board by using an add-in board of FMC (FPGA Mezzanine Card). In case of two-dimensional mesh network, if one cluster is configured in one board (as shown in Fig. 5), a total of four links are sufficient for all directions (bi-directional communication) to connect to neighboring clusters, thereby a communication network can be constructed by using these add-in boards in addition to the main boards. As a result, it is theoretically possible to build the evaluation platform for many-core with infinite number of cores, by using the communication interface mentioned above.

In this evaluation platform, the clock frequency of processor core is assumed to be 10 MHz. Since the serial data communication between routers based on rocket I/O can be performed up to 5 Gbps, the bandwidth of data transmission is capable of ensuring sufficient bandwidth for the core clock even with control information in each packet (currently, 32-bit data and 4-bit control). However, in an aurora module it takes about 180 ns of communication latency for serial synchronization. This indicates two additional processor clock cycles in the communication latency as compared to that of the case where routers are directly connected within a board. Though this could lead to degrade the accuracy of the evaluation, the overall effect on the evaluation cycle is trivial because the on-chip network is typically used for the main memory and I/O accesses which need much longer latency.

3.4 Prototype System

The photographic view of our developed evaluation platform is shown in Fig. 6. Currently, sixteen ML605 boards are connected for forming the 4×5 two-dimensional mesh network where sixteen core clusters and one peripheral cluster are provided. Because the peripheral cluster is located on the same FPGA device implementing one of the core clusters, the number of total clusters is seventeen even with sixteen ML605 boards, and the network forms the structure of 4×5 mesh. Since the basic organization of a core cluster has 8 cores, there is a total of 128 processor cores in this evaluation platform. Thanks to symmetry of the mesh network, the configuration of each core cluster is exactly the same, and thereby, it is easy to extend the platform to a larger environment with more than 128 cores.

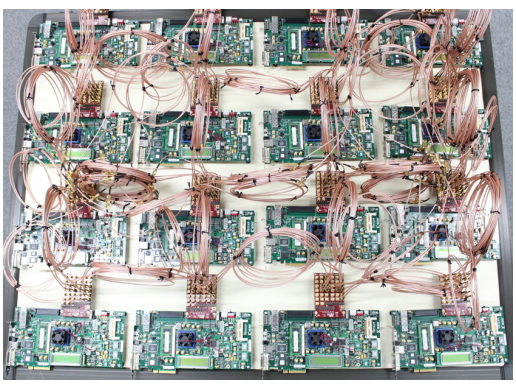

Fig. 6 Photographic view of our developed evaluation platform.

4. Evaluation

4.1 Experimental Settings

In order to evaluate SMYLEref architecture and to verify its prototype, we conducted experiments on our developed evaluation platform with several parallel programs. Because the distributed shared L2 cache on multiple clusters is under development, the L2 cache is disabled for evaluations with multiple clusters. When the evaluation is conducted within a cluster, the L2 cache is used. We will note whether the L2 cache is used or not in the paper for each evaluation.

The clock frequency for each functional block is set as follows.

- Geyser core: 10 MHz
- Cluster internal bus, router, and peripheral bus (PLB): 5 MHz
- DDR3-SDRAM: 100 MHz

To verify the basic functionality and to perform preliminary evaluation for SMYLEref, we use parallel applications from a popular parallel benchmark suite SPLASH2 for our initial experiment. FFT and LU programs are employed for the experiment. We use the pthread version of those benchmark programs. For code generation, we use gcc 4.4.6 targeted for MIPS processors.

FFT and LU contain floating-point operations. We could not implement 8 cores on the FPGA device if an FPU is integrated in each core as described in Sect. 3.2. Therefore, we perform the evaluation in two cases; one is 8 cores per cluster configuration without FPUs which can evaluate up to 128 cores on 16 FPGA boards, and the other is 4 cores per cluster with FPUs which can evaluate up to 64 cores. For the case without the FPUs, we use gcc's soft-float option in which floating-point arithmetic operations are done by software emulation.

4.2 Parallel Processing API

As mentioned in the previous subsection, we use pthread API for parallelizing the evaluated programs. We develop the simple version of pthread library with a minimum set of pthread functions for SMYLEref evaluation environment. By using this pthread library, the evaluated programs are compiled and executed without any code modifications except for option parsing. The current developed pthread functions include functions for creating and joining threads (*pthread_create, pthread_join*, etc.), mutex lock operations (*pthread_mutex_lock, pthread_mutex_unlock*, etc.), and functions for barrier synchronization (*pthread_barrier_wait* etc.).

Table 3 Area Overhead.

	Slices (%)	Flip-Flops (%)	LUTs (%)
Core cluster			
Geyser core	3,301 (8.76%)	7,089 (2.35%)	10,942 (7.26%)
L2 cache	2,853 (7.57%)	6,232 (2.07%)	8,813 (5.85%)
Router	1,170 (3.11%)	838 (0.28%)	3,400 (2.26%)
MEM access controller	201 (0.58%)	338 (0.11%)	533 (0.35%)
Board comm. controller	1,257 (3.33%)	3,059 (1.01%)	2,775 (1.84%)
FPU	3,703 (8.80%)	15,968 (5.21%)	15,526 (10.30%)
Peripheral cluster			
I/O controller	1,596 (4.24%)	6,007 (1.99%)	2,577 (1.71%)
Router	1,170 (3.11%)	838 (0.28%)	3,400 (2.26%)
Packet controller	147 (0.39%)	168 (0.05%)	240 (0.16%)

The inter-cluster cache coherence control is outside of the scope of SMYLEref as well as our evaluation environment. Besides, cache coherence is essentially required for FFT and LU programs since SPLASH2 is a benchmark for shared memory computer systems. Therefore, we use some tricks to obtain the correct execution results in those programs. First, the dynamically allocated data region (heap data) is allocated to the uncachable virtual address domain which is not accessed through a cache. Second, L1 data cache is flushed in each core every time the barrier synchronization or the mutex lock function is called. By this way, the consistency of the cached data can be guaranteed without the need of modifying the program codes.

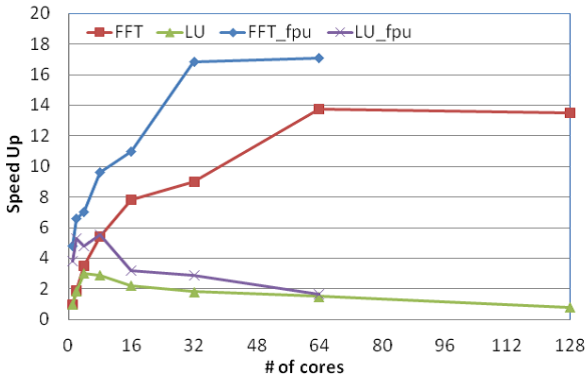

Fig. 7 Evaluation results of parallel speedup.

4.3 Implementation and Evaluation Results

The resource usage of the FPGA device for the major modules such as the Geyser core, the router, and the board communication controller are listed in Table 3. The implementation results show that, the Geyser core consumes the largest amount of hardware resources in comparison with that of the other modules (except the FPU). It is approximately three time larger than the router or board communication controller. L2 cache takes 2,853 slices which are slightly smaller than that of the Geyser core. The area overhead of MEM access controller or packet controller is very small as compared to the area overhead of the Geyser core. In the case of configuring eight cores in a core cluster, most of the hardware resources are used for the Geyser cores, and area overhead of the other modules is not significant. This indicates that clustering by several cores contributes to the reduction of area overhead due to the router for NoC. If the FPU is implemented in a core, it will become the largest consumer of hardware resources.

The evaluation results of FFT and LU benchmark programs in terms of parallel processing efficiency are shown in Fig. 7. In the figure, FFT and LU indicate the results without FPUs (utilizing the soft-float function), whereas FFT_fpu and LU_fpu express the results which are obtained by using the hardware FPUs. The results are normalized to the performance of one core without FPUs for each program. As seen from the figure, the performance increases in accordance with the increase in the number of cores in FFT. The performance is saturated at 64 cores. However, the speed-up is not adequate as compared to the number of used cores. This is mainly caused by memory access bottleneck. Since the heap memory area is uncachable, most of the data accesses goes to the main memory which causes the cluster bus and network congestions. The situation is worse in LU. Very small performance improvement or even worse performance degradation is observed with the increase in the number of cores. This is due to the memory access bottleneck as well as cache flushing needed when the synchronization and the mutex lock functions are called.

Specially in the case of LU_fpu, a slight performance degradation is observed in 4 cores compared with 2 cores and 8 cores in Fig. 7. This is due to the traffic congestion in the cluster bus. The data traffic of a cluster bus becomes the heaviest in the case of 4 cores which is the maximum number of available cores in a core cluster when hardware FPUs are used. When 8 cores are used, the traffic load of a cluster bus gets slightly smaller since another cluster joins the computation.

Note that the final target of this research is using many-cores as accelerators by a parallel programming model which does not require the cache coherence mechanism such as OpenCL. If parallel program written in that model is executed on this environment, we will see much higher parallel performance. The evaluation with such programing model is our future work.

Typically, the main memory access speed is very slow as compared to the processor speed. However, in our evaluation environment, clock frequency of the processor core is 10 MHz, whereas frequency of main memory SDRAM

is 100 MHz. The relative DRAM access time is much faster than the realistic systems. Therefore, we implement a DRAM access latency control mechanism by which we can set an arbitrary DRAM access latency in the environment. Figure 8 shows the performance results of FFT varying additional DRAM access latency. FFT1, FFT5, FFT10, FFT50 in the figure indicate the results when the memory access is delayed by 1, 5, 10 and 50 processor clock cycles, respectively. As shown in the figure, performance is degraded as memory access latency increases, resulting in worse parallel processing efficiency. We should notice here that, we can evaluate the performance of target programs with variable important parameters such as main memory latency. This is the benefit of our evaluation platform.

Generally, the parallel processing efficiency can be enhanced by the use of the L2 cache. Moreover, if a cache coherence mechanism is provided, data in heap memory region can be accessed through the cache and no cache flush is necessary, thereby better parallel processing efficiency can be achieved. Figure 9 shows the performance of the evaluated programs when an L2 cache and cache coherence mechanism are used within a cluster. Note that because a cluster has only 8 cores, we can evaluate up to the 8-core case in this evaluation. FFT_L2&Coh and LU_L2&Coh represent the evaluation results when enabling the L2 cache and the cache coherence. As shown in this figure, a great improvement of parallel speedup is observed in both programs. This indicates that making good use of caches is very important.

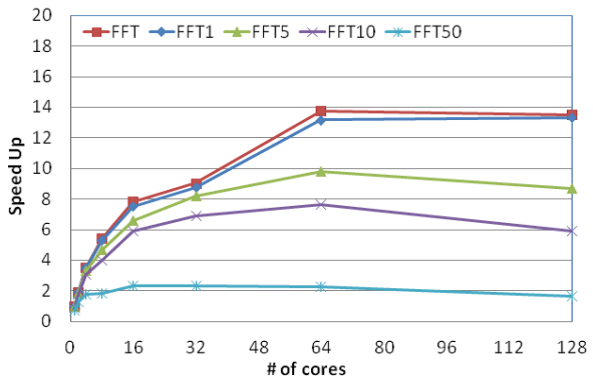

Fig. 8 Influence of memory access time on performance.

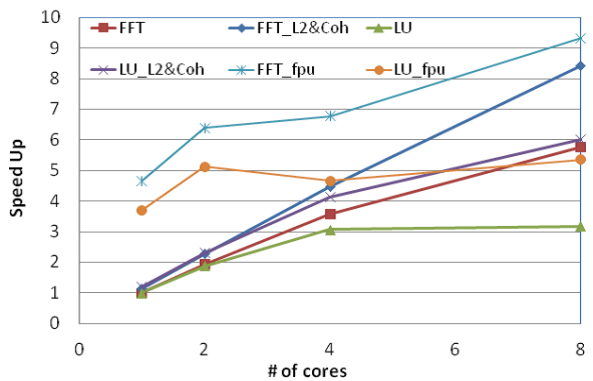

Fig. 9 Performance improvement with L2 cache and coherence.

In our prototype system, the default L2 cache access latency takes 7 processor clock cycles. In fact, it would be much slower than 7 processor cycles. In order to provide a capability to evaluate several cases for L2 cache access latency, we also implement a L2 cache access latency control mechanism. The evaluation results when the L2 cache access latency is delayed by 1, 5, 10, and 20 processor clock cycles for FFT with the cache coherence mechanism (indicated by FFT_L2&Coh1, FFT_L2&Coh5, FFT_L2&Coh10, and FFT_L2&Coh20, respectively) are shown in Fig. 10. From the figure, no big performance difference is observed among different cases of L2 cache access latency. In 8 cores, the speedup difference between 7-cycle (FFT_L2&Coh) and 27-cycle (FFT_L2&Coh20) of L2 access latency is only 0.17%. Since all the data are accessed through the caches and the L1 data cache captures most of the data accesses, L2 cache access is not so frequent. In fact, the ratio of total L2 cache access time to total execution time is about 1% even in FFT_L2&Coh20. Therefore, L2 cache access latency does not affect the performance very much in this program.

As described in Sect. 3, the benefit of prototyping with FPGA is evaluation speed and scalability. The comparison of evaluation time (or simulation time) between our platform and software simulator is presented in Fig. 11. We used MARSS simulator [11] as the software simulator. The MARSS simulator is know as one of the fastest cycle accurate simulator with capability of full system simulation. Though the architecture model of SMYLEref is different from that used in MARSS, we set the architecture parameters such as issue width and cache configuration for MARSS to get a model, which is similar to our evaluation platform as much as possible. The following is the specification of the host machine which the MARSS simulator is performed on.

- CPU: Intel core-i7 X980, 3.33 GHz.
- LLC (L3): 12 MB.
- Main Memory: DDR3-1066, 6 GB.

In the Fig. 11, FFT_soft and LU_soft indicate the simulation time in software simulator. Because of the limitation of the simulator, we can evaluate only up to 8 cores. The

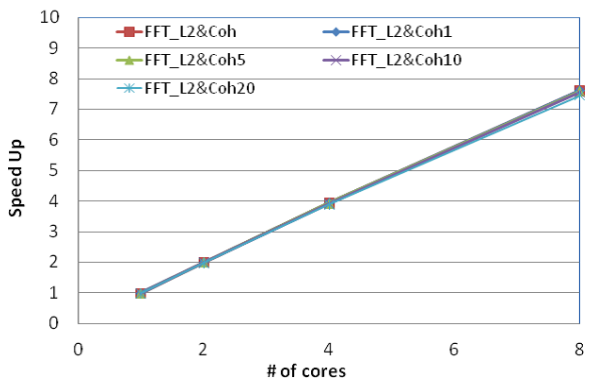

Fig. 10 Influence of L2 cache access time on performance.

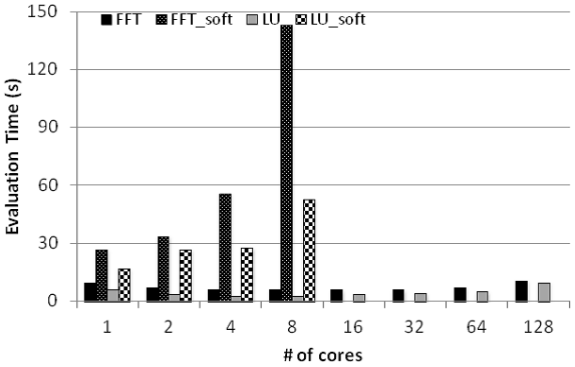

Fig. 11 Evaluation time between our platform and software simulator.

results marked as FFT and LU are the evaluation time in our platform. As shown in the figure, the simulation time in the software simulator for 1-core is slightly longer than our evaluation environment. Though in our evaluation environment, the evaluation time gets shorter in the increase of the number of used cores, the required time for simulation in the software simulator becomes very longer. In the case of 8-core, the evaluation time of software simulator is 20 times slower than that of our platform. This indicates that the prototyping with FPGA has very good scalability. From this result, we can conclude that the FPGA prototyping is a very promising option for evaluating many-core processors with hundreds of cores.

5. Related Work

In recent years, there have been many researches and developments of processor architecture and compiler for multi-core and many-core. The OSCAR multi-core processor and the automatic parallelization compiler are introduced in [13] and [14]. Besides, an actual multi-core chip with the ability of reducing power consumption by an automatic parallelizing compiler has also been developed [15]. A coherence control mechanism by compiler for many-core processors is proposed in [16]. These researches and the actual product do not adopt an NoC based structure which will be essential aspect of many-core processors. Since the SMYLEref evaluation platform can evaluate an NoC based many-core processor with more than a hundred of cores, it is very beneficial for evaluating future many-core processors.

As reported in [17], the authors introduce an emulation environment named *ScalableCore system* for scalability, flexible and high-speed many-core processor, which is formed from multiple small-capacity FPGA boards. In addition, a number of evaluation platforms using FPGA, such as RAMP [18] are also developed for research on multi-core and many-core processors. The target of the SMYLEref evaluation platform is shared memory many-core systems with hundreds of cores. In this point, our work is differ from those FPGA based platforms.

Nowadays, the experimental products as well as commercial products of the many-core processor with several tens cores are becoming practical. Intel Inc. introduces a processor with 80 cores for experiment, which one tera FLOPS or more can be attained at a chip [19]. The floating point cores in this processor is built in the 8×10 two-dimensional mesh network. ATAC processor [20] is a many-core processor built from 1024 cores, that can provide a high-speed global broadcasting network using an on-chip optical network. A scalable directory-based cache coherence protocol called *ACKwise* using the above-mentioned optical network is implemented for ATAC processor. In addition, the TILE64 of Tilera Inc. [21] has been also provided as a commercial many-core processor. TILE64 combines 8×8 homogeneous cores using a mesh on-chip network.

Since these processors are implemented in the real LSI chips, they require very high cost of development and implementation. Moreover, once the actual chips are fabricated, it is impossible to change the hardware parameters such as the memory access latency. On the other hand, our platform is very flexible so that some of the parameters are easy to modify as well as wide variety of statistical information (the number of cache misses etc.) is easy to observe. In this point, the benefit of our evaluation platform is emphasized.

6. Conclusion

In this paper, we introduce SMYLEref, a many-core architecture actualizing the concept of virtual accelerator on many-core (VAM), to realize a high performance and low power many-core processors for embedded systems. The prototype system for SMYLEref with FPGA evaluation boards is also presented. In addition, to evaluate SMYLEref architecture and verify its prototype, the initial experiments with the use of several SPLASH2 benchmark programs are conducted on our developed 128-core evaluation platform. The evaluation results are affirmed and the observations on them are also discussed.

In the future, we consider implementing and developing the inter-cluster cache coherence mechanisms, practical architectures for many-core, and the software development environment. Besides, the evaluations of various realistic applications for embedded systems are also taken into account in our future work.

Acknowledgements

We would like to gratefully acknowledge the worthy discussions and advise of Prof. Mitaro Namiki of Tokyo University of Agriculture and Technology and all the people of Namiki lab. during this work. A part of this research is supported by NEDO (New Energy and Industrial Technology Development Organization) project for Extremely Low-power Circuits and Systems (Green IT Project) and JSPS KAKENHI Grant Number 24680004.

References

[1] H. Nakamura, H. Amano, K. Usami, M. Namiki, M. Imai, and M. Kondo, "A plan of innovative power control for ultra low-power

and high-performance system LSIs," IEICE Technical Report, ARC-173-14, 2007.
[2] N. Seki, et al., "A fine grain dynamic sleep control scheme in MIPS R3000," IEICE Trans. Inf. & Syst.(Japanese Edition), vol.J93-D, no.6, pp.920–930, June 2010.
[3] I. Mogi, K. Kimura, T. Sunata, and M. Namiki, "Porting Linux OS to an evaluation board for power-saving MIPS processor "Geyser"," IPSJ Technical Report, OS-114-9, 2010.
[4] K. Usami and N. Ohkubo, "A design approach for fine-grained run-time power gating using locally extracted sleep signals," Proc. International Conference on Computer Design (ICCD), pp.155–161, 2006.
[5] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," IEEE Trans. Commun., vol.47, no.8, pp.1260–1267, 1999.
[6] S.T. Nguyen and S. Oyanagi, "A high-throughput router architecture with on-the-fly virtual channel allocation for on-chip networks," IPSJ Trans. ACS, vol.4, no.2, pp.84–93, 2011.
[7] Li-S. Peh and W.J. Dally, "A delay model and speculative architecture for pipelined routers," Proc. 7th International Symposium on High-Performance Computer Architecture (HPCA), pp.255–266, 2001.
[8] W.J. Dally and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2004.
[9] N. Binkert, et al., "The gem5 simulator," ACM SIGARCH Computer Architecture, vol.39, no.2, pp.1–7, 2011.
[10] H. Kasture, et al., "Graphite: A distributed parallel simulator for multicores," Proc. International Symposium on High-Performance Computer Architecture (HPCA), pp.1–12, 2010.
[11] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A Full System Simulator for Multicore x86 CPUs," Proc. Design Automation Conference (DAC), pp.1050–1055, 2011.
[12] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," Proc. 22nd International Symposium on Computer Architecture (ISCA-95), pp.24–36, 1995.
[13] H. Kasahara, H. Honda, A. Mogi, A. Ogura, K. Fujiwara, and S. Narita, "A multi-grain parallelizing compilation scheme for OSCAR (optimally scheduled advanced multiprocessor)," Lect. Notes Comput. Sci., vol.589, pp.283–297, 1992.
[14] K. Kimura, Y. Wada, H. Nakano, T. Kodaka, J. Shirako, K. Ishizaka, and H. Kasahara, "Multigrain parallel processing on compiler cooperative chip multiprocessor," Proc. 9th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9), pp.11–22, 2005.
[15] M. Ito, et al,, "An 8640 MIPS SoC with independent power-off control of 8 CPU and 8 RAMS by an automatic parallelizing compiler," Proc. IEEE International Solid State Circuits Conference (ISSCC 2008), 2008.
[16] M. Mase, K. Kimura, and H. Kasahara, "Parallelizing compiler directed software coherence," IPSJ Technical Report vol.2010-ARC-189-7, pp.1–10, 2010.
[17] S. Takamaeda, S. Watanabe, K. Kyou, N. Fujieda, and K. Uehara, "Scalable Core system: Hardware environment for many-core architectures evaluation," IPSJ Trans. ACS, vol.4, no.1, pp.24–42, 2011.
[18] RAMP - Research Accelerator for Multiple Processors, http://ramp.eecs.berkeley.edu/
[19] S.R. Vangal, et al.: An 80-tile sub-100-W teraFLOPS processor in 65-nm CMOS, IEEE J. Solid-State Circuits, vol.43, no.1, pp.29–41, 2008.
[20] G. Kurian, et al., "ATAC: A 1000-core cache-coherent processor with on-chip optical network," 19th International Conference on Parallel Architectures and Compilation Techniques (PACT'10), pp.477–488, 2010.
[21] S. Bell, et al., "TILE64 - Processor: A 64-core SoC with mesh interconnect," Proc. IEEE International Solid State Circuits Conference (ISSCC 2008), pp.588–598, 2008.

Son Truong Nguyen received his B.Sc. degree in Electrical Engineering from Le Quy Don Technical University, Vietnam in 1996 and his M.E. degree in Department of Mathematics and Computer Science, National Defense Academy, Japan in 2004. He received the Ph.D. degree in Graduate School of Information Science and Engineering, Ritsumeikan University, Japan in 2011. He is currently a researcher at Graduate School of Information Systems, the University of Electro-Communications, Japan. His research interests focus on interconnection networks, high-performance computing and multi/many-core architectures.

Masaaki Kondo received the B.E. degree in College of Information Sciences and the M.E degree in Doctoral Program in Engineering from University of Tsukuba in 1998 and 2000 respectively, and the Ph.D. degree in Graduate School of Engineering from the University of Tokyo in 2003. He is currently an associate professor in the Graduate School of Information Systems at the University of Electro-Communications in Tokyo. His research interests include computer architectures, high-performance computing, and dependable computing. He is a member of the IEEE, the ACM, and the IPSJ.

Tomoya Hirao was born in Hyogo prefecture, Japan in 1977. He received the B.E. from Wakayama University, Japan in 2000 and M.E. from Nara Institute of Science and Technology, Japan in 2002. In 2011, he joined Department of Advanced Information Technology, Kyushu University, Japan. Currently, He is an engineer of the Fixstars Corporation, Japan.

Koji Inoue was born in Fukuoka, Japan in 1971. He received the B.E. and M.E. degrees in computer science from Kyushu Institute of Technology, Japan in 1994 and 1996, respectively. He received the Ph.D. degree in Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan in 2001. In 1999, he joined Halo LSI Design & Technology, Inc., NY, as a circuit designer. He is currently an associate professor of the Department of Advanced Information Technology, Kyushu University. His research interests power-aware computing, high-performance computing, dependable processor architecture, secure computer systems, 3D microprocessor architectures, and multi/many-core architectures.