Development of a Many-core Architecture for Automotive Embedded Systems

Trung-Dung Pham, Van-Tien Nguyen, and Truong-Son Nguyen
Department of Control Engineering, Le Quy Don Technical University, Hanoi, Vietnam
Email: thchdung@yahoo.com, {tiendiep73, sonhuongjp}@gmail.com

***Abstract*—Nowadays, the number of cores in a micro-processor used for automotive embedded systems are trending to be increased from tens (multi) to hundreds (many) of cores to achieve higher performance. Developing many-core architectures with high-performance and low-power is becoming a major technical challenge. In this paper, we propose a solution to develop the many-core architecture by using many low-performance but small and very low-power cores to obtain very high performance and efficient parallel processing. The many-core evaluation platform is implemented on FPGA utilizing multiple mass-produced evaluation boards, which can result to several advantages such as low cost of development, flexibility and easy implementation as compared to that of manufacturing in a real LSI chip. In addition, some evaluation results of parallel benchmark programs conducted on our developed platform are also presented and discussed.**

***Index Terms*—many-core architecture, evaluation platform, network-on-chip, parallel benchmark, FPGA**

I. Introduction

Over the last ten years, automotive embedded systems have taken a leap in engineering. The amount of digital information a car has to compute in real-time is growing exponentially. Multiple tasks/operations on a car such as powertrain and chassis control (engine, steering, brake …), body electronics (instrument panel, lighting, air bag …), multimedia applications (car navigation, traffic information, electronic toll collection …) as well as integrated services (electronic stability control, pre-crash safety, parking assistance …) are required to be processed in parallel with high reliability and safety.

To meet above-mentioned requirements, lots of multi-core architectures of micro-processor with tens to hundreds of processor cores on a single chip has been proposed and developed for automotive embedded systems. In advance, a prospect of many-core architecture which consists of hundreds to thousands of cores is being expected for higher performance. Therefore, developing many-core architectures and compiler techniques to realize a high performance and low power many-core processor used in automotive engineering is becoming a major technical challenge.

Up to now, many multi/many-core processor architecture and compilers are researched and developed for the system-on-a-chip designs. A multi-core processor named OSCAR and its automatic parallelization compiler is introduced in [1] and [2]. Besides, an actual multi-core chip with the ability of reducing power consumption by an automatic parallelizing compiler is also developed [3]. A coherence control mechanism by compiler for many-core processors is proposed in [4]. However, these architectures do not adopt a NoC-based structure which is a very significant aspect of many-core processors. In practice, it is hard to extend to hundreds of cores in such a bus-based architecture.

Thanks to the advance of technology, nowadays, many experimental products as well as commercial products of the NoC-based many-core processor with several tens to hundreds of cores are becoming practical. In 2008, Intel Inc. has introduced a processor with 80 cores for experiment, which one tera FLOPS or more can be achieved [5]. The floating point cores in this processor are built in the 8x10 two-dimensional mesh network. In 2010, a many-core processor called ATAC [6] built from 1024 cores that can provide a high-speed global broadcasting network using an on-chip optical network is proposed. A scalable directory-based cache coherence protocol called *ACKwise* using the optical network is developed for improving ATAC processor performance. At the same time, the TILE64 of Tilera Inc. [7] has been provided as a commercial many-core processor. In TILE64, 8x8 homogeneous cores are organized in a mesh on-chip network called iMesh.

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 hardware parameters such as the memory access latency, the number of cache misses etc. In addition, in case of changing the number of cores, another LSI die is required to be mounted.

In our research, we propose a solution to develop the many-core architecture by using many low-performance but small and very low-power cores to obtain very high performance and efficient parallel processing. Our goal is to realize a low-power cost effective embedded processor as an alternative for the current system-on-a-chip designs in automotive area.

Generally, in order to research many-core architectures, developing an evaluation platform is very important. Usually, software simulators can be used for architecture

Manuscript received October 7, 2014; revised May 11, 2015.

doi: 10.12720/joace.4.2.147-152

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 very 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 a solution of developing a many-core evaluation platform using multiple mass-produced FPGA evaluation boards, which can result to several advantages such as low cost of development, flexibility and easy implementation as compared to that of manufacturing in a real LSI chip. In addition, several initial experiments with parallel benchmark programs on our developed platform are also discussed.

The remainder of paper is organized as follows. Section II describes the detail of our proposed many-core architecture. The construction of evaluation platform developed on FPGA is presented in Section III. Some initial benchmark evaluation results are discussed in Section IV. Finally, Section V concludes the paper.

II. Proposed Many-core Architecture

The block diagram of our proposed many-core architecture is shown in Fig. 1. In this architecture, multiple blocks called *clusters* are organized 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. 1) 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 shared by all cores in the chip.

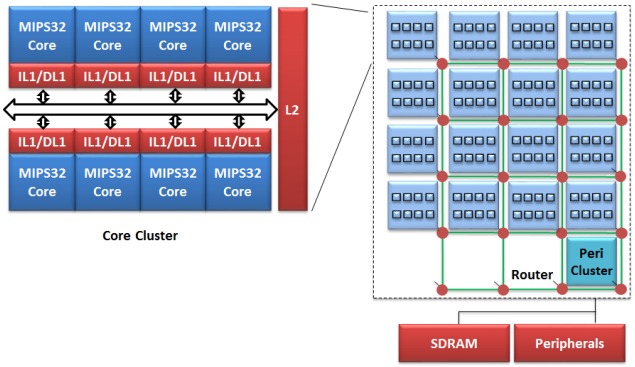

Figure 1. Proposed many-core architecture.

In this architecture, we use a simple scalar processor core based on the MIPS32 architecture [8]. The processor core is designed to allow launching an operating system such as the Linux. This processor core consists of a simple 5-stage pipeline, 8KB L1 instruction and L1 data caches, and a 16-entry TLB (Translation Look-aside Buffer) which can be controlled by instructions. The data cache employs the write-back policy.

A. Cluster Organization

Our proposed many-core architecture is constructed by two types of clusters: a *core cluster* with a number of the above-mentioned processor cores, and a *peripheral cluster* which allows to interface with the external devices. In the core cluster, multiple simple scalar processor cores are connected through a cluster bus. A router for the network-on-chip is also connected to this bus. A peripheral cluster consists of several controllers for SDRAMs, external I/Os, and a router. The access to the main memory and I/O devices is ensured by the packet-based data transfer, which is performed between the core cluster and the peripheral cluster through an on-chip network.

B. Network-on-Chip

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. 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 [9].

For data transfer, the virtual-channel flow control [10] is used. The packets are divided into multiple flits (flow control digits) where a two-flit header (i.e. the header includes two flits) contains all necessary information for routing and memory access including a route for the next hop (the channel selection signal for the succeeding router), a cluster ID, a core ID, and memory access controls (read/write, non-burst/burst, etc.). The other flits of the packet carry payload data. The format of a two-flit header is illustrated in Fig. 2, where *FID* and *VCID* indicate the flit ID and virtual channel ID, respectively. As shown in the figure, each cluster is distinguished by a unique *cluster ID*, and each core within a cluster can be identified by a *core ID*.

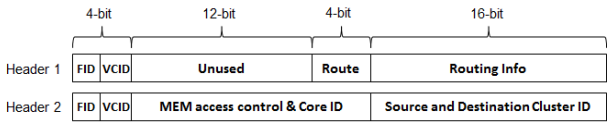

Figure 2. Two-flit header description.

To form the on-chip network, we utilize a low cost single-cycle router [11], which is an enhanced design of the conventional virtual channel (VC) router architecture. This router can perform high performance data transfer in a single cycle with low hardware cost which is able to contribute to energy-efficiency.

The block diagram of the router used in our many-core architecture is shown in Fig. 3. 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.

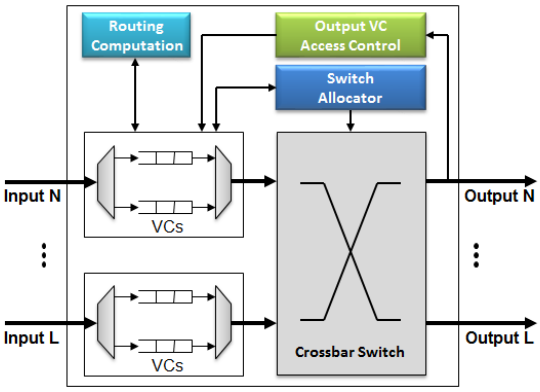

Figure 3. Single-cycle router architecture.

III. Many-core Arcitecture Implementation on FPGA

In development and verification of architectures and software for many-core processors, the evaluation platform plays a very important role. Lots of software simulators have been used for many-core architecture research and exploration. However, the issue of a quite long time of simulation for hundreds of cores is becoming a critical. An alternative to the software simulator which can overcome the above problem is an emulator implemented on FPGA. The solution of using FGPA allows to improve the speed of evaluation considerably as compared to that of software simulators, while still maintaining an acceptable cost of development. In addition, it is not difficult to develop additional hardware modules at register transfer level (RTL) to extend the evaluation environment.

Figure 4. ML605 evaluation board.

For the mentioned-above remarkable advantages, we decided to choose the solution of using FPGA for the implementation of our many-core evaluation platform. The Xilinx ML605 evaluation board (Fig. 4) 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 Virtex-6 device are summarized in Table I. We use Verilog-HDL for the circuit design and the Xilinx ISE for the Verilog-HDL simulation and implementation to the FPGA.

TABLE I. Specifications of ML605 Evaluation Board and Virtex-6 Chip

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

A. Core/Peripheral Cluster Implementation

The overview of a core cluster implementation is depicted in Fig. 5. Each core cluster is formed from several processor cores (up to eight cores) and a single-cycle router as described in Section II.B. In addition, the L2 cache is also located on the core cluster. The L1 data/instruction caches, L2 cache 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 processor 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 other 2-bit is used for the flit identification of the packet and the remainder is needed for managing buffer credits of the router. The memory access caused by L1 cache misses is controlled by a module named *MEM Access* Controller. If the address of a memory access operation 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 global main memory via the router.

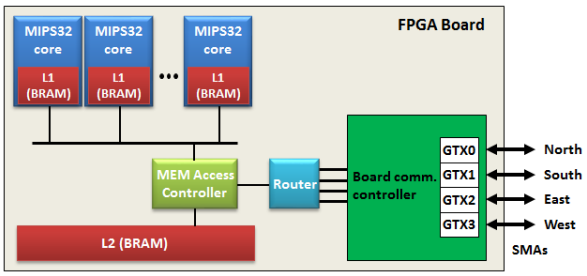

Figure 5. Block diagram of a core cluster.

The peripheral cluster is composed of several controllers of peripherals such as the SDRAM, UART, and SysACE which are combined by the 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 the 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 platform enables to set the additional memory access latency arbitrarily, which provides a contribution to the evaluations on the impact of memory access latency.

B. Off-board Communications

Since the hardware resource of the FPGA device (Virtex-6 XC6VLX240T) used for our evaluation platform is only enough for about eight processor cores, a

many-core processor with tens or hundreds of cores cannot be implemented on one ML605 board. Hence, our many-core 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 quite small, it can be configured in the same board in accordance with one of the core clusters (Fig. 6).

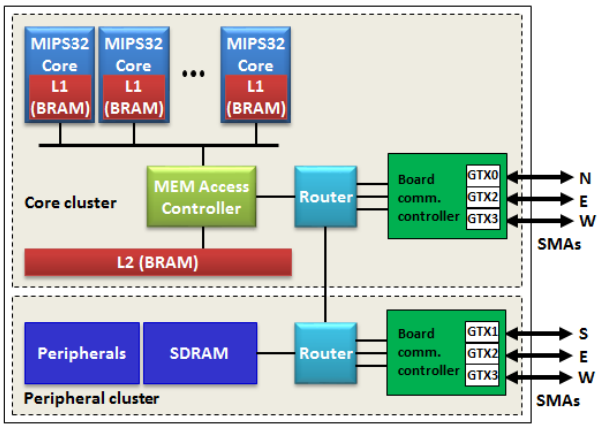

Figure 6. Configuration of a peripheral cluster and a core cluster in one board.

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. The rocket I/O communication 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, theoretically it is possible to build the evaluation platform for many-core with infinite number of cores, by using the communication interface mentioned above.

In our current platform, eight ML605 boards are connected to form a 4x3 two-dimensional mesh network where eight 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 numbers of total clusters are nine and the size of network is 4x3. A core cluster consists of 8 cores, therefore a total of 64 processor cores are available in the platform. Thanks to the symmetry of a mesh network, it is possible to extend the platform to a larger environment with more than 64 cores easily.

C. Implementation Results

The resource usage of the FPGA device for the major modules such as the processor core, router, and board communication controller are listed in Table II. The implementation results show that, the processor core consumes the largest amount of hardware resources in comparison with that of the other modules. It is approximately three times larger than the router or board communication controller. The area overhead of MEM access controller or packet controller is very small as compared to the area overhead of the processor core. In the case of configuring eight cores in a core cluster, most of the hardware resources are used for the processor 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.

TABLE II. AREA OVERHEAD

	Slices (%)	Flip-Flops (%)	LUTs (%)
Core cluster			
Processor core	3,301 (8%)	7,089 (2%)	10,942 (7%)
Router	1,170 (3%)	6,232 (2%)	8,813 (6%)
MEM controller	201 (0.6%)	338 (0.1%)	533 (0.3%)
Board controller	1,257 (3%)	3,059 (1%)	2,775 (2%)
Peripheral cluster			
I/O controller	1,596 (4%)	6,007 (2%)	2,577 (2%)
Router	1,170 (3%)	838 (0.3%)	3,400 (2%)
Packet controller	147 (0.4%)	168 (0.1%)	240 (0.2%)

IV. BENCHMARK EVALUATION

A. Experimental Settings

In order to evaluate the proposed many-core architecture, we conducted experiments on our developed evaluation platform with several parallel programs. At this point, the L2 cache coherence mechanism is under development, therefore the L2 cache is disabled for evaluations with multiple clusters. The clock frequency for each functional block is set as follows: 20MHz for processor core, 10MHz for cluster bus, and 100MHz for DDR3-SDRAM.

To verify the basic functionality and to perform preliminary evaluation, we use parallel applications from a popular parallel benchmark suite called SPLASH2 [12] for our initial experiment. We employ FFT and LU programs which are widely used in digital signal processing and image processing for the experiment. We use the *pthread* version of FFT and LU benchmark programs. For code generation, we use gcc 4.4.6 targeted for MIPS32 processors.

For the FFT and LU which contain floating-point operations, we choose the solution of using gcc's soft-

float option in which floating-point arithmetic operations are done by software emulation.

To use *pthread* API for parallelizing the evaluated programs, we develop a simple version of *pthread* library with a minimum set of *pthread* functions for our many-core evaluation environment. By using this *pthread* library, the evaluated programs are compiled and executed without any code modification except for option parsing. Currently, basic functions for creating/joining threads, mutex lock operations, and barrier synchronization are provided in our developed *pthread*.

B. Evaluation Results

The evaluation results of FFT and LU benchmark programs in terms of parallel processing efficiency are shown in Fig. 7. The results are normalized to the performance of one core for each program. The figure shows that, performance of FFT increases in accordance with the increase of the number of cores. However, the speed-up is not adequate as compared to the number of used cores. The main reason of this circumstance is the memory access bottleneck. Since the heap memory area is uncachable, most of the data access operations go to the global main memory which causes the cluster bus and network congestions. The situation is worse in LU. A slight improvement of performance or even worse performance degradation is observed with the increase of 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.

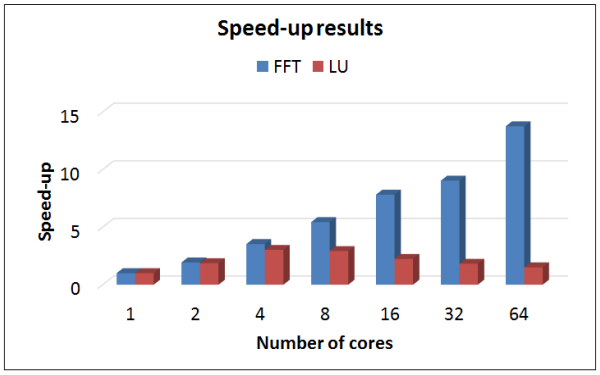

Figure 7. Evaluation results of parallel speed-up.

In general, the parallel processing efficiency can be improved by the use of the L2 cache. Moreover, if a reasonable control mechanism for cache coherence is provided, data in heap memory region can be accessed through the cache and no cache flush is necessary. As a result, a significant enhancement of parallel processing efficiency can be achieved.

Obviously, the solution of developing a many-core architecture based on low-performance but small and very low-power cores can help attaining very high performance and efficient parallel processing, without the complexity of implementation.

As mentioned in Section III, the advantages of developing many-core evaluation platform on FPGA are evaluation speed and scalability. The comparison of evaluation time (or simulation time) between our platform and software simulator is presented in Fig. 8. We use MARSS simulator [13] as the software simulator. The MARSS simulator is known as one of the fastest cycle accurate simulator with capability of full system simulation. Though the model of our proposed many-core architecture is different from that used in MARSS, we set the architecture parameters such as issue width and cache configuration to get a suitable one, which is similar to our architecture as much as possible. The specifications of host machine for MARSS simulator are shown as follows.

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

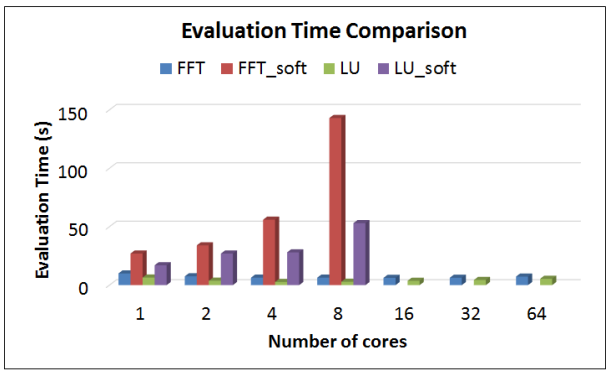

Figure 8. Evaluation time between our platform and software simulator.

In the Fig. 8, FFT_soft and LU_soft indicate the simulation time in software simulator. Because of the limitation of MARSS simulator, we can measure the simulation time for maximum of 8 cores. The 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 one core is slightly longer than our evaluation environment. When the number of used cores increase, the evaluation time gets shorter in our evaluation environment, while the required time for simulation in the software simulator becomes very longer. In the case of eight cores, the evaluation time of software simulator is 20 times slower than that of our platform. This indicates that the FPGA emulation has a very good scalability. From these observations, we can conclude that implementing many-core platform on FPGA is an appealing alternative for evaluating many-core processors with several tens to hundreds of cores.

V. Conclusion

In this paper, we introduce a many-core architecture for realizing a high performance and low power many-core processors for automotive embedded systems by using many low-performance but small and very low-power cores. The solution of implementing the proposed architecture with multiple mass-produced FPGA evaluation boards is presented. In addition, for the purpose of evaluating performance and parallel processing efficiency of the proposed many-core architecture and verifying its hardware system, the initial experiments with the use of several SPLASH2

benchmark programs are conducted on our developed 64-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 as well as the software development environment. Besides, the evaluations of various realistic applications for automotive embedded systems are also taken into account in our future work.

REFERENCES

[1] 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)," *Lecture Notes in Computer Science*, vol. 589, pp. 283-297, 1992.

[2] K. Kimura, Y. Wada, H. Nakano, T. Kodaka, J. Shirako, K. Ishizaka, and H. Kasahara, "Multigrain parallel processing on compiler cooperative chip multiprocessor," in *Proc. 9th Workshop on Interaction between Compilers and Computer Architectures*, 2005, pp. 11-22.

[3] M. Ito, *et al.*, "An 8640 mips soc with independent power-off control of 8 cpu and 8 rams by an automatic parallelizing compiler," in *Proc. IEEE International Solid State Circuits Conference*, 2008.

[4] M. Mase, K. Kimura, and H. Kasahara, "Parallelizing compiler directed software coherence," *IPSJ Technical Report*, vol. 2010-ARC-189-7, pp. 1-10, 2010.

[5] S. R. Vangal, *et al.*, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 29-41, 2008.

[6] G. Kurian, *et al.*, ATAC, "A 1000-core cache-coherent processor with on-chip optical network," in *Proc. the 19th International Conference on Parallel Architectures and Compilation Techniques*, 2010, pp. 477-488.

[7] S. Bell, *et al.*, "Tile64 - processor: A 64-core soc with mesh interconnect," in *Proc. IEEE International Solid State Circuits Conference*, 2008, pp. 588-598.

[8] MIPS Technologies, Inc. (2001-2002). Mips32 architecture for programmers. [Online]. Available: http://www.imgtec.com/mips/architectures/mips32.asp

[9] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260-1267, 1999.

[10] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[11] S. T. Nguyen and S. Oyanagi, "An improvement of router throughput for on-chip networks using on-the-fly virtual channel allocation," in *Proc. the 24th International Conference on Architecture of Computing Systems*, 2011, pp. 219-230.

[12] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. the 22nd International Symposium on Computer Architecture*, 1995, pp. 24-36.

[13] A. Patel, F. Afram, S. Chen, and K. Ghose, "MArss: A full system simulator for multicore x86 cpus," in *Proc. Design Automation Conference*, 2011, pp. 1050-1055.

Trung-Dung Pham was born in Hanoi, Vietnam in 1963. He received his B.E. degree in control engineering from Le Quy Don Technical University, Vietnam in 1986, and M.E. degree in space engineering from SUPAERO, France in 1998. He received the Ph.D. degree in computer system, network and software form Moscow State Aviation Technological University, Russia in 2004. He is currently an associate professor and is the dean of the department of control engineering, Le Quy Don Technical University. His research interests include high-performance computing, multi/many-core architectures, digital signal processing and automotive control systems.

Van-Tien Nguyen was born in Vinh Phuc province, Vietnam in 1973. He received his B.E and M.E degrees in control engineering from Le Quy Don Technical University, Vietnam in 1998 and 2006, respectively. He is currently a lecturer of the department of control engineering, Le Quy Don Technical University. His research interests include computer architectures, automation control systems and image processing.

Truong-Son Nguyen was born in Vinh Phuc province, Vietnam in 1974. He received his B.E. 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 was a postdoctoral fellow at Graduate School of Information Systems, the University of Electro-Communications, Japan, between 2011 and 2013. He is currently a lecturer of the department of control engineering, Le Quy Don Technical University. His research interests focus on interconnection networks, high-performance computing, multi/many-core architectures and automotive control systems.