4J-2

Design and Implementation of Manycore Processor for a Large FPGA

Haruka MORI † Shimpei SATO ‡ Chu Van Thiem ‡ Kenji KISE ‡
Dept. of Computer Science, Tokyo Institute of Technology †
Graduate School of Information Science and Engineering, Tokyo Institute of Technology ‡

1 Introduction

Due to the limitations of the single-threaded performance and the improvement of semiconductor technology, the number of cores integrated on a single chip continues increasing. As can be seen in Intel Xeon Phi[1] and KARLAY MPPA MANYCORE[2], manycore processors are becoming mainstream.

In this paper, with the above background, we design and implement a simple manycore processor that operates as an accelerator targeting a large FPGA (Field Programmable Gate Array). Additionally, we describe an overview of the designed manycore processor 's architecture and the amount of used hardware resources in the FPGA implementation.

2 Design of Manycore Processor for a Large FPGA

2.1 Design Guide

In this work, we assume that our manycore processor is implemented on a Virtex-7 FPGA board. However, since various models of FPGAs and FPGA boards with different specifications are being manufactured, the portability of the implementation is important. Therefore, we keep to a minimum the use of non-FPGA hardware in our manycore processor design.

The Virtex-7 FPGA board has two types of RAM resources: On-Chip RAM and Off-Chip DRAM. The On-Chip RAM called BRAM (Block RAM) is the internal memory of the FPGA. And the Off-Chip DRAM is integrated on the FPGA board. By utilizing the Off-Chip DRAM together with BRAM, the available memory amount is increased. However, the use of Off-Chip DRAM make the implementation heavily depend on the specification of the board. To prevent the degradation of the board-to-board portability, we only use BRAMs in our implementation.

We assume that our manycore processor is implemented on the Virtex-7 FPGA, and is used by connecting to a host computer. At the start of the operation, the user transfers a binary code which can be executed by the processor to the FPGA via USB serial communication using a terminal emulator such as Tera Term. The execution result of the processor is transferred back to the host computer via the same USB serial communication, and is displayed on the terminal emulator.

A manycore processor generally has multiple PEs (Processor Elements). It is possible to improve throughput of a process in manycore processors by running a parallel application. Although running parallel applications on the processor is one of our design goals, we only focus on designing the processor to operate reliably on the FPGA at this stage. Therefore, we do not assume that parallel applications run on the processor.

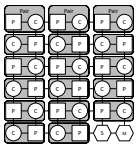

Figure 1: Architecture of a manycore processor with 36 nodes configurationn

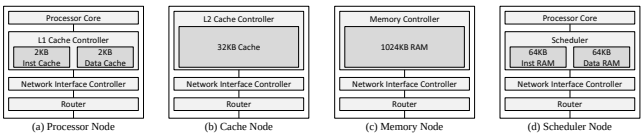

Figure 2: Internal structures of four types of nodes

2.2 Architecture

Figure 1 shows architecture of a manycore processor with 36 nodes configuration. There are four types of nodes: Processor Node (P), Cache Node (C), Scheduler Node (S), and Memory Node (M). These 36 nodes are connected with two-dimensional mesh network. The node-to-node communication is performed by the packet exchange. In **Figure** 1, the solid lines indicate the physical links between the nodes. Each gray colored area shows a pair of Processor Node and Cache Node. By pairing the Processor Node with the Cache Node, the Cache Node becomes the private cache of the Processor Node.

Figure 2 shows the internal structures of four types of nodes. As illustrated in Figure **Figure** 2a, a Processor Node is composed of one processor core, 2KB instruction cache and 2KB data cache. The direct mapped structure is employed as the data storage structure of the instruction cache and the data cache. The processor core is a pipelined MIPS processor[3] with 32-bit data path. The five pipeline stages include instruction fetch, instruction decode, execute, memory access, and write back. A Cache Node has a 32KB cache as shown in **Figure** 2b. As same as the caches at the Processor Nodes, we use the direct mapped structure for the caches at Cache Nodes. A Memory Node (**Figure** 2c) has a 1MB RAM. This 1MB RAM is divided into 16 areas of 64KB. Programs transferred from the host computer are sequentially stored in these memory areas. We will describe Scheduler Node (**Figure** 2d) in the next section.

2.3 Scheduler Node

The designed manycore processor stores all programs in the memory at the beginning of its operation. The Scheduler Node has a role of allocating memory areas in which programs are stored to the Processor Nodes. If the number of programs that have been transferred is greater than the number of processor nodes, when execution of one program at a Processor Node finishes, the Scheduler Node allocates a new program to that Processor Node.

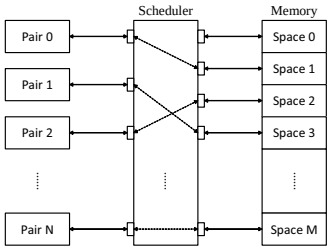

Figure 3: An example of the allocation by the scheduler

```
1  int main(void){
2    int pair, space=0;
3
4    for(pair=0; pair < N+1; pair++){
5      activate(pair, space);
6      space += 1;
7    }
8
9    while(space < M+1){
10     if(get_stat()){
11       pair = find_deactive();
12       activate(pair, space);
13       space += 1;
14     }
15   }
16
17   return 0;
18 }
```

Figure 4: C program describing an allocation algorithm

Figure 3 shows an example of the allocation by the scheduler. In this example, there are N pairs of Processor Node - Cache Node, and M memory areas in which programs are stored. The Scheduler Node allocates space 1 to pair 0, space 3 to pair 1, space 2 to pair 2, and space M to pair N. If pair 0 has completed the execution of the program, the Scheduler Node can allocate space 0 to pair 0.

The allocation algorithm is described by users. Like the Processor Nodes, the Scheduler Node consists of one processor core, 64KB instruction memory, and 64KB data memory. The allocation program is transferred from the host computer and stored in the RAM of the Scheduler Node at the start of the processor 's operation.

Figure 4 shows a C program describing an allocation algorithm. In this example, we assume that $N < M$ where N is the number of pairs and M is the number of memory areas. Lines 4-7 describe the initialization sequence. The Scheduler Node sequentially allocates a memory area for each pair. Lines 9-15 describe the additional sequence. In this sequence, the Scheduler Node sequentially allocates memory areas for pairs that finished their operations.

The Scheduler Node can estimate performance based on the distances between the pairs and the Memory Node. Further, the number of fetches from the Processor Nodes can be obtained. Even if a program is repeatedly executed by different pairs, an allocation algorithm that allocates the program on a priority basis to the pairs that exhibit better performance can be applied.

3 Implementation and Evaluation

In the implementation, we use the VC707 Evaluation Board which integrate the Virtex-7 XC7VX485T FPGA. The Virtex-7 XC7VX485T FPGA has 75,900 slices. The minimum processor configuration has four nodes: A Processor Node, a Cache Node, a Memory Node, and a Scheduler Node. These four nodes are connected in a 2×2 two-dimensional mesh network. We described the logic circuit of the processor in Verilog HDL, and performed synthesis in Xilinx ISE 14.5. We have implemented the processor on the VC707 Evaluation Board. And the behavior of some simple programs and scheduling programs has been confirmed.

Table 1: Amount of hardware and logic synthesis time

	Slices	Slice Regs	Slice LUTs	BRAM	TIME
2x2	4,508	6,146	13,692	305	1,262
4x4	27,045	31,813	75,391	377	2,983
6x6	51,108	75,058	175,309	497	6,163

Table 1 shows the amount of hardware and a logic synthesis time of different nodes configurations. In addition to the Scheduler Node and the Memory Node, 2×2 consists of a pair of Processor Node Cache Node, 4×4 consists of 7 pairs, and 6×6 consists of a 17 pairs.

In the routers of the nodes at the edge of the mesh network, there are several unused ports. Since these redundant logics are optimized by ISE, the amount of hardware is reduced. In the case of 2×2 mesh network, the number of slices is reduced significantly because all nodes are positioned at the corners. In the case of 4×4 and 6×6, the nodes that are close to the center of the mesh network are not affected by the optimization. When the impact of optimization is small, the number of slices of a Processor Node and a Cache Node is about 1800 and 2000, respectively. The increment of the number of slices compared to the optimized case is a reasonable.

On the other hand, BRAMs are used for implementing caches, memories. And the number of used BRAMs scales with the number of Processor Nodes and Cache Nodes.

4 Summary and Future Work

In this paper, we designed a simple manycore processor targeting a large FPGA. We implemented the designed manycore processor in the Virtex-7 FPGA and measured the amount of hardware.

As future work, we plan to improve the architecture and evaluate performance of the designed manycore processor using different scheduling algorithms and parallel applications.

References

1) Intel Xeon Phi Coprocessor: Datasheet. `http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-datasheet.html`.

2) KALRAY MPPA MANYCORE. `http://www.kalray.eu/products/mppa-manycore/`.

3) David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 2011.