2A-06

Enabling Fast Thousand-Core Processor Emulation using FPGAs

Thiem Van Chu † Kenji Kise †

† Graduate School of Information Science and Engineering, Tokyo Institute of Technology

1 Introduction

The microprocessor industry has shifted to integrating multiple processor cores on a chip due to the power wall and the limitation of instruction-level parallelism techniques. Multi/many-core processors have now become main stream. Several processors with up to 100 cores are already on the market. To keep continuous improvements in system performance, the core count is expected to grow up to 1000 cores and beyond.

To design novel many-core processors with hundreds to thousands of cores, simulation environments play a decisive role. Unfortunately, conventional software simulators scale poorly. Their simulation speed typically varies from several to hundreds of Kilo Instructions Per Second depending on the level of detail of the simulation models, and therefore the simulation of one second of one core may take several days to several months to complete. Simulating n cores is fundamentally over n times slower than simulating one core due to the communication overheads. Thus, providing a practical simulation time for designing large-scale many-core processors is a challenging problem. Although some approaches such as using only a subset of benchmarks and parallelizing the software simulators leveraging modern substrates (e.g., multi-core PCs, clusters) have been proposed to tackle the low simulation performance, none of them can significantly improve simulation speed without sacrificing simulation accuracy.

In this study, we use FPGAs, which have been adopted in accelerating many computational tasks such as data processing in data centers, for fast and accurately emulating many-core processors with hundreds to thousands of cores. We describe several techniques to overcome the capacity constraints of FPGAs while leveraging their high degree of parallelism to improve the simulation speed.

2 FPGA-Accelerated Simulation

The many-core processor simulation problem can be tackled by exploiting the fine-grained parallelism of FPGAs. In our FPL 2015 paper [1], we propose some novel techniques for emulating large-scale Network-on-Chip (NoC) designs on a single FPGA. However, designing a novel many-core processor requires fast and accurate simulation of not only the on-chip network but also the cores and memory.

When the FPGA-based approach is adopted, there are two problems that must be resolved. The first problem is that it is hard to scale the simulation model to support large designs due to the FPGA capacity constraints. For example, even a very large FPGA has only around 10MB of on-chip memory which is far from enough for implementing the inside states of all cores and caches of a many-core processor. The second problem is the difference between the final ASIC and FPGA. The simulated processor cores may contain some modules that are inefficient to implement directly on FPGA. For example, a Content-Addressable Memory (CAM) or a memory with more than two ports are widely known to be FPGA-inefficient structures. They would consume a large amount of resources and operate at low frequencies if being implemented directly on FPGA.

3 Proposed Approach

3.1 Time-Multiplexing

We adopt the time-multiplexing technique to scale our simulation model to support designs with hundreds to thousands of cores. In this paper, we focus on techniques on an FPGA. To efficiently leverage multiple FPGAs to emulate larger designs, some extensions are required, but left as future work.

The time-multiplexing technique has been adopted by some existing FPGA-based processor emulators including HAsim [2]. However, these emulators use only one physical processor core and one physical NoC router to sequentially emulate all cores and routers of a processor, and therefore their simulation speed decreases dramatically as the number of simulated cores is increased up to hundreds to thousands. Our insight is that, many-core processors with direct interconnection networks such as k-ary n-cubes can be emulated using multiple interconnected physical nodes, each is composed of a processor core and a NoC router.

Figure 1(a) shows an example where a 4×4 tile processor is emulated by using two physical nodes. We call the group of interconnected physical nodes physical cluster. Figure 1(b) shows the high-level datapath when the time-multiplexing technique is adopted. As discussed in Section 2, even a very large FPGA has only a limited amount of on-chip memory. Thus, we use off-chip DRAM in addition to the on-chip memory to be able to store the inside states of all logical clus-

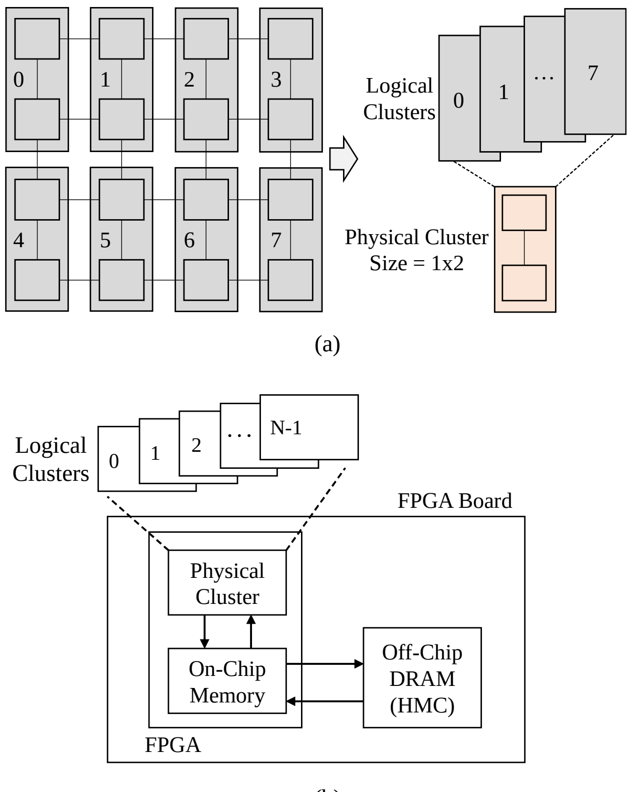

Figure 1: (a) A 4×4 tile processor is emulated by using two physical nodes. (b) High-level datapath when the time-multiplexing technique is adopted.

ters, the data passed between the logical clusters, and caches. The bottleneck of off-chip memory access can be overcome by using Hybrid Memory Cube (HMC), an emerging high-performance DRAM interface supported in many modern FPGA boards.

3.2 Decoupling Processor Clock from FPGA Clock

As discussed earlier, simulating a processor on FPGAs is not trivial because some modules of the processor may be FPGA-inefficient structures. Directly implementing these structures on FPGAs would result in a low operating frequency and a large amount of FPGA resource consumption. If we decouple the processor clock from the FPGA clock, we can use multiple FPGA clocks to emulate a clock of the FPGA-inefficient structures. By this way, we can choose an implementation that efficiently utilizes FPGA resources while achieving a high operating frequency. Although the FPGA-cycles-to-processor-cycles ratio is greater than 1, the emulator performance may not be affected because of the improvement in operating frequency. For example, a memory with three read ports and three write ports can be efficiently implemented on FPGA using embedded SRAMs (block RAMs in Xilinx FPGAs) if we use three FPGA cycles to emulate a cycle of the memory because an embedded SRAM typically has only one read port and one write port. The direct implementation is slow because it uses discrete registers and large multiplexers. If the operating frequencies of the direct implementation and the implementation with three FPGA cycles per emulation cycle are 50MHz and 200MHz, respectively, the indirect implementation can achieve even better performance.

By decoupling the processor clock from the FPGA clock, we can also easily implement large memories such as caches using both on-chip and off-chip RAM. For example, when a cache is implemented in this way, if a processor core reads a cache line that is not stored in the on-chip memory, we need several FPGA cycles to fetch the cache line from the off-chip memory.

With the above approach, each module of a processor may take a different number of FPGA cycles to complete its one cycle. Thus, we need to synchronize the operations of all modules inside the processor. In a naive synchronization method, before proceeding to the next cycle, every module has to wait for the module that takes the most FPGA cycles to finish its operation of the current cycle.

However, since the time-multiplexing is used to sequentially emulate the entire processor using a physical cluster, we can divide the processing of each module inside the physical cluster into multiple stages and execute these stages in a pipeline fashion. In particular, at any FPGA cycle in a processor cycle, each pipeline stage can be executed by a different logical cluster. This can be performed because, in a processor cycle, there is no data dependence among all logical clusters.

4 Conclusion

In this paper, we described an approach for enabling high-speed simulation of many-core processors with hundreds to thousands of cores using FPGAs. We show that, by efficiently using the time-multiplexing technique and decoupling the processor clock from the FPGA clock, the simulation model can be scaled to support large designs. Our future work is to actually implement and evaluate the proposed approach on FPGA boards.

References

[1] T.V. Chu, S. Sato, and K. Kise, ”Ultra-Fast NoC Emulation on a Single FPGA”, in Proceedings of the 25th International Conference on Field-Programmable Logic and Applications (FPL 2015).

[2] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, ”HAsim: FPGA-Based High-Detail Multicore Simulation Using Time-division Multiplexing”, in Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA 2011).