A Novel Hybrid Pipeline Design Methodology on a Multi-Cores Streaming System for Multimedia Applications

Yu-Chi Su, Sung-Fang Tsai, You-Ming Tsao, and Liang-Gee Chen
DSP/IC Design Lab, Graduate Institute of Electronics Engineering
National Taiwan University, Taipei, Taiwan
{steffi,bigmac,eddie,lgchen}@video.ee.ntu.edu.tw

***Abstract*—With the rapid growth of media-processing technologies and the advancement of semiconductor process, more and more multimedia applications are integrated into consumer electronics. However, in such highly complex system, the design time for the circuit designers does not reduce much as the process advances. We propose a novel hybrid pipeline design methodology for multi-core streaming system from C-level to RTL design. Our methodology can optimize data communication for media-processing applications to achieve both flexibility and efficiency.**

I. Introduction

Recently, numerous multimedia applications including video/audio/image processing and 3-D graphics are embedded in modern consumer electronics such as PDA, smart phones and so on. An application specific integrated circuits (ASIC), designed with optimized data paths for one special multimedia application, can achieve high performance for a specific application. However, such dedicated hardware accelerators lack flexibility and result in high design effort. On the other hand, traditional general-purpose programmable processors, which benefit from design flexibility and fast time-to-market, attempt to support a wide range of applications so that meeting performance requirements in real-time when processing multimedia applications are nearly impossible. Stream processors, designed for media-processing applications, fills the gap between special-purpose ASICs and traditional programmable processors to achieve both efficiency and flexibility [1-2]. However, despite the optimized architecture of a stream processor for media-processing applications, a multi-core stream processor system still suffers from performance degradation due to bus contention, resulting in limited scalability. Therefore, a complete and application-driven design methodology addressing communication issues to develop a scalable multi-core stream processor system and supporting designers ease design space exploration at system-level is needed. In this paper, we propose a novel hybrid pipeline design methodology on multi-core streaming system from c-level down to RTL. The goal of the design methodology is to develop a multi-core streaming system for multimedia applications in both efficiency and flexibility.

II. The Stream Processing Model

Media-processing applications have the following characteristics: data parallelism and high computation-to-memory ratio. The stream programming model exploits these properties to express a media application as a stream program: The abundant and repeated input data can be viewed as a sequence of stream elements. A stream element is a group of user-defined data structure. For example, a stream element can be a single pixel from an image, a 256 pixels macroblock in video encoder, or a vertex data in graphics. Stream data is a finite sequence of user-defined data element and processed through a series of computation kernels.

The stream programming model divides an application into two parts: data communication and data computation. In a streaming system, data access unit is separated from data computation units.

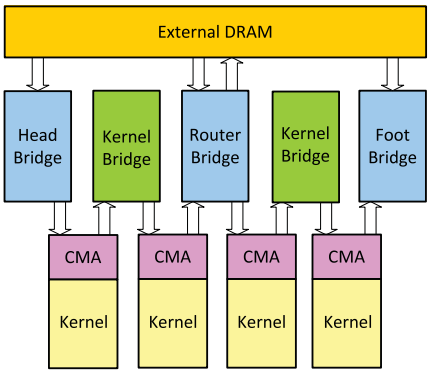

Fig. 1. Multi-Core stream processor architecture

Data access units dealing with data movement concurrently operate with computation units to speed up processing time. Kernel, the computation unit in a stream processor, exploits the parallelism of a stream program with multiple execution units that can process multiple stream elements in parallel. To capture data locality of a stream program that has higher immediate data locality within kernel and lower data access rate to the external memory, the memory system in a stream processor is designed as a hierarchical architecture with degrade bandwidths from kernel to the external memory. With exploiting parallelism and locality of stream programs, stream processors can meet high performance. Moreover, the stream processor is programmable that helps designers to reduce development time. Due to the significant improvement of performance and flexibility, the stream processor has become the main stream for modern high-computing devices, such as GPUs.

III. System Architecture

This section introduces the architecture of the proposed multi-core streaming system. Fig. 1 shows the system block diagram. The streaming system is designed based on the stream programming model. The primitive units are unified stream cores and bridges, representing data computation units and data access units in the system, respectively. Since the data access unit is separated from the computation unit, repeated data movement and data processing can operate at the same time. Bridges implement data access behaviors for streaming system. We design four kinds of bridges with different functions- head bridges, kernel bridges, route bridges, and foot bridges. Kernel bridges gather or split stream elements from the last stage of kernel and then send new structured stream elements to the kernel at the next stage. The head bridge manages data movement from the external memory and the foot bridge moves data in the opposite direction. Route bridges can both send/receive data to/from the memory. Behaviors of all kind of bridges are defined by users on design time to optimize communication in efficiency.

Besides bridges, kernels are implemented as unified stream cores. A stream core can execute multiple threads at the same time to

978-1-4244-6673-3/10/$26.00 ©2010 IEEE

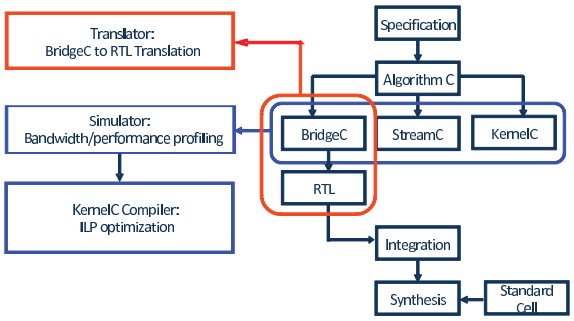

Fig. 2. The New Design Flow.

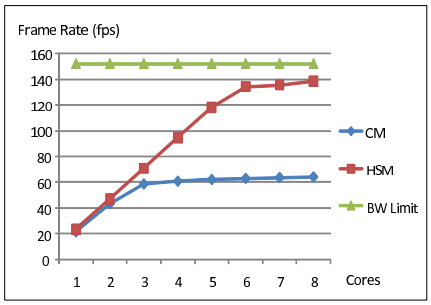

Fig. 3. Performance Scalability.

provide the maximum parallel processing capability in instruction level and thread level. Moreover, the stream processor platform can also achieve task-level parallelism by chaining multiple stream cores to execute a series of procedures in pipeline. A modified configurable memory array (CMA), which can be configured as cache memory or buffer, is served as a local store for each stream core.

When the system starts, the head bridge receives stream load commands and configurations of data accessing pattern. After that, the head bridge begins to fetch stream elements, such as macroblocks (MB) for video coding from the external memory and then send data to CMA of a kernel. As to kernels, they load stream elements from the previous stage of bridge and then process multiple stream elements in parallel. The stream elements are processed through a series of procedures on kernels. Finally, the output stream elements are generated and stored back to the external memory. In the process, the series of bridges and kernels operate concurrently and iteratively until all of the stream element have been processed. The overall systems runs the target application in pipeline manner. All the media algorithm can be mapped in a pipeline fashion. The designers need to determine how many stages are enough.

IV. Overview of The New Design Flow

Fig. 2 shows our new design methodology for VLSI design in hybrid design style. After a C-algorithm has been designed and verified, it will be partitioned into three major parts: KernelC, StreamC, and BridgeC. StreamC is a serious high-level API to set configuration information to the head bridge. The configuration information shows the head bridge how to access data from the external memory and rearrange data for the kernel at the next stage. BridgeC is an user-defined fix function that can move data between CMA and the external memory and restructure data for the next kernel. In our flow, BridgeC will be translated to dedicated circuits by BridgeC-to-RTL translator. In this way, the data communication part of an multimedia algorithm, which is always the performance bottleneck of a multi-core system, can be customized to be a specific hardware circuits to achieve high efficiency. KernelC is the computation part of the multimedia algorithm. We have developed a KernelC-compiler that compiles KernelC to generate kernel instructions with instruction-level parallelism(ILP) optimization. Kernel instructions are finally mapped onto the stream cores. In this way, a stream core can be programmed with different functions according to various applications with high flexibility.

In our work, a multi-core streaming simulator has also been developed to support system-level analysis for designers to make design choices, like number of cores, different data accessing methods, and any other important factors that significantly affecting system performance. The simulator can help exploring design space and optimizing performance for the input C-algorithm. The simulator can either run in transaction level modeling(TLM) or cycle-accurate level(CA). In TLM mode, the kernelC is directly linked with the simulator for fast simulation. Designers can do functional verification on C-level, estimate memory bandwidth and examine approximate performance. Besides, performance bottleneck can be discovered and resolved in this stage. In CA mode, the kernelC is compiled and optimized to kernel instructions. The simulator will run in hardware scheduling and simulate system performance. In this mode, cycle-accurate performance and instruction-level verification can be achieved.

V. Simulation Results

We use ConvergenSC to build a system-level platform, including an ARM968EJS core, external memory systems, OCP bus, and our multi-core stream processor simulator. We take motion compensation in SVC [3] decoder as the software for case study. First, the software code of SVC is implemented as StreamC, KernelC and BridgeC and then we follow the proposed design methodology shown in Fig. 2 to generate the corresponding design and run it on our simulator.

We compare performance for the motion compensation module in the SVC decoder between two implementations with different data access methods: (1)hardware-managed streaming memory model and (2)cache-based memory model, called HSM and CM, respectively, for short. HSM, based on the proposed design flow, implements the data access unit of the target design with specific circuits. As to CM, it adopts cache to access data from the external memory. Fig. 3 shows that HSM outperforms than CM in scalability with respect to number of stream cores. The comparison between the two methods measures the impact of the data access method for multi-core stream processors.

VI. Conclusion

We propose a novel and scalable design methodology for multi-core streaming system. To release the high design effort of ASICs and heavy communication of traditional programmable cores, the proposed design flow provides a fast and flexible method to create multi-core streaming system that can efficiently process media applications. This design methodology also includes a system-level analysis method to help designers mapping software and exploring design space. Experiment results show that the created target in our methodology can achieve performance scalability.

References

[1] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson and J. D. Owens "Programmabel stream processor," In *IEEE Computer*, Aug. 2003, vol. 36, no. 8, pp. 54V62.

[2] U. J. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany Kapasi and et. al., "The Imagine Stream Processor," In *Proceedings 2002 IEEE International Conference on Computer Design*, Sep. 2002, pp. 282–288.

[3] H. Schwarz, D. Marpe, T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard ," In *IEEE Trans. Circuits Syst. Video, Technol.*, Vol. 17, No. 9, pp.1103-1120, 2007.