A device driver for the AB21 SoC in Supreme-K system

Youngho Kim, Eunji Lim, Shinyoung Ahn, Yoomi Park
Supercomputing Technology Research Center
Electronics and Telecommunications Research Institute
Daejeon, South Korea
{kyh05,ejlim,syahn,parkym}@etri.re.kr

***Abstract*— This paper presents an XPU device driver (XDD) that efficiently supports the OpenCL programming model and development tools for the AB21 SoC, a heterogeneous computational accelerator used in the homegrown prototype of the Supreme-K supercomputer. We propose a dynamic polling-based termination detection mechanism for parallel computing kernel tasks to improve the scheduling performance of OpenCL work groups in a split driver model architecture for heterogeneous compute accelerators connected to the host system via PCIe. Furthermore, we enhance the architecture of XDD to support debugging through the hardware DM of the RISC-V ISA core. This enhancement facilitates the efficient development and optimization of various parallel computing applications.**

Keywords—AB21 SoC, XPU, Heterogeneous Accelerator, OpenCL, Device driver

I. INTRODUCTION

Computational accelerators, such as GPUs, TPUs, or FPGAs, have been widely utilized in recent HPC systems. Besides, research on the development of heterogeneous SoC hardware, constructed with both a general-purpose processor core and heterogeneous computation accelerators, is in active progress. GPUs and other accelerators are broadly employed in high-performance computing fields, including artificial intelligence, big data, and supercomputers [1,2,3,5]. Companies such as Nvidia, AMD, and Google broadly employ conventional computation accelerators primarily in the HPC industry.

Customized optimization is limited in the heterogeneous accelerator and processor-based high-performance computing system environment of a new architecture that integrates processor cores and accelerator cores from various hardware and system vendors due to the separation of hardware design and software stack implementation. In contrast, during the hardware design and development stage of system processors and nodes, efficient parallel computing applications can be developed by carrying out a co-design process that closely involves software stacks, including high-level programming models, compilers, runtimes, and device drivers. It is possible to develop an optimized software stack for performance and supporting customized programming models. Computational accelerator devices like Nvidia GPU, Google TPU, and AMD HAS/Rock offer not only dedicated software stacks optimized for their own accelerators for parallel application execution but also a range of development support tools, such as debuggers and profilers. Efficient utilization of new computational accelerator devices and development of device-optimized parallel applications require the provision of development support tools, such as profilers and debuggers, for application and system program developers.

In Supreme-K project, which is to develop supercomputer prototype based on home-grown massively parallel processing unit. Fig. 1 shows the HW architecture and SW stacks of Supreme-K system. The core technology of the Supreme-K supercomputer, which is a self-developed SoC that integrates a commercial ISA core (ARM armv 8.4) and a parallel accelerator in a single chip as a cache coherent mesh network, and aims to achieve FP64 computational theoretical performance of 16 TFLOPS. OpenCL was adopted as a programming model to take advantage of SoC's parallel computing capability and named SOCL (Supreme-K OpenCL). The development of parallel accelerator ISA-based SOCL compiler, runtime, and device driver is in progress [2].

There are three key techniques we have in place to efficiently support the execution and development of parallel program applications on the AB21 SoC device.:

- We propose a scheduling algorithm using the split driver model for workgroup scheduling. The algorithm is based on scheduling metadata stored in shared memory on a SoC device, which allows for the simultaneous execution of parallel computational kernel tasks.
- We propose a workgroup scheduling method and termination detection mechanism for kernel execution based on dynamic periodic polling. Our approach utilizes a device-side dedicated core that supports efficient scheduling of parallel computing kernel tasks.
- We offer the device driver architecture and function extensions required for debugger and profiler tools to support parallel computing applications on AB21 SoC devices for performance optimization.

In this paper, we describes the architecture and core design of the XPU device driver (XDD) that enables optimized execution of the parallel computation kernel on the AB21 SoC device. The AB21 SoC device is a heterogeneous core-based computational accelerator located in the computation node of the Supreme-K supercomputer system. It also elaborates on the modification of the architecture to support debugger, profiler, and management tools. These tools provide a platform for developing and verifying the upper layer's software stacks. Finally, it concludes by explaining the future direction of development and plans for the AB21 SoC device driver. This device driver enables efficient simultaneous execution of multi-user processes.

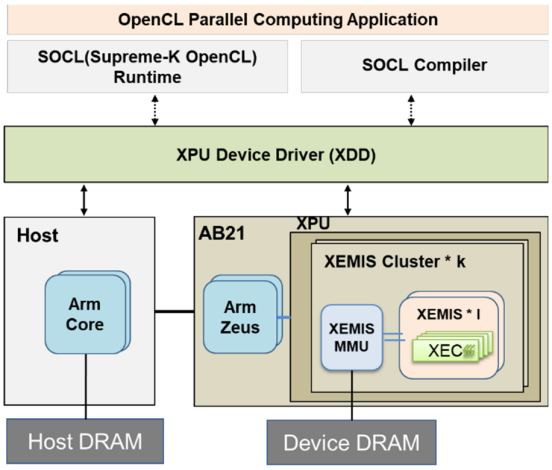

Fig. 1. Supreme-K System Architecture diagram

II. Overview of XDD development on AB21 SoC

A. Supreme-K System architecture

The development HW platform used in this paper is the AB21 SoC, which is a self-developed, heterogeneous accelerator SoC. It integrates a commercial ISA core and XPU (eXtreme Processing Unit) - the core technologies of the Supreme-K supercomputer prototype - into a single chip connected using a cache coherent mesh network (CMN) [2]. Each XPU has a cluster connected to a single SMMU configured for every eight parallel computation units (XEMIS) to facilitate fast address conversion. Additionally, a single AB21 has an ARM SMMUv3's complex device architecture with SMMUs embedded in four clusters. Each XEMIS executes a workgroup of the OpenCL parallel computation kernel allocated by the scheduler, and parallel computation is processed exclusively as a HW thread through the Computation Accelerator Unit Module (XECM) that supports RISC-V ISA.

B. XPU Device Driver Development Overview

For concurrent design, development, and operational verification of the AB21 compiler, SOCL runtime, and other components, we used XDE (XPU Driver Emulator) and AB21Q for software and hardware emulation. We created an emulation platform called XPUSim and XDE (XPU Driver Emulator) for fast prototyping, which facilitated the concurrent development of the compiler, OpenCL backend, and SW stack based on Runtime for XPU devices[4]. Furthermore, we developed the prototype HW and SW components before the SoC development, which consisted of porting, integration testing, XDD, upper SW stack applications, and system software, through a collaborative design approach across the hardware and software development teams on an FPGA-based AB21 HW platform system. However, during this fast prototype development, the parallel application program's performance degraded due to the scheduling overhead caused by the allocation of work-groups for XEMIS of the XPU unit and the interrupt-based kernel task termination notification.

As shown in Figure 2, when a running kernel terminates, the status register in each XEMIS unit of the XPU changes from XEMIS_BUSY to XEMIS_IDLE and a hardware interrupt is generated. The interrupt handler requests scheduling from the workgroup scheduler provided by the XDD. The entire scheduler's response time for scheduling is when an interrupt occurs to indicate the termination of kernel execution, and the workgroup scheduler assigns a new workgroup to the corresponding register and requests it to be executed. XEMIS, as a parallel computation unit allocated for executing parallel computation kernel tasks within the XPU, has representative methods for notifying the termination of OpenCL work group execution through interrupt and polling mechanisms. The interrupt-based mechanism for notifying the result of task execution requires a relatively high processing cost for the interrupt service handler, due to context switching. This mechanism is dedicated to scheduling work-groups in the HW scheduler, for example, Nvidia GPU and terminates kernel execution for all work-groups. It is common to process post-work completion notification only as an interrupt. The status register-based polling mechanism for WG execution uses a dedicated core and can quickly ascertain the end of execution of each WG while supporting fast scheduling. During the processing of large-scale computations that take a long time to execute in a work-group, a problem of system load increase due to incessant polling may arise.

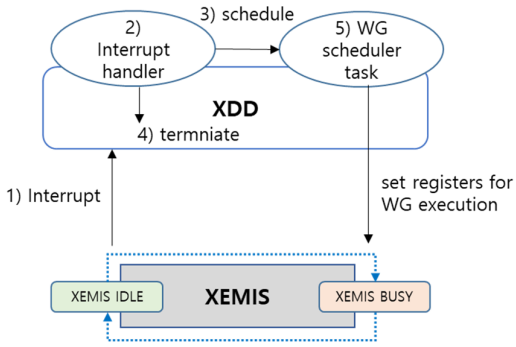

Fig. 2. Flow diagram of Interrupt-based notification of kernel task termination and Work-group scheduling on XPU

III. XPU Device Driver Design

To develop better device drivers that support the OpenCL model and development tools such as the debugger on the AB21 SoC, we are using heterogeneous computing acceleration on our own prototype supercomputer. Our proposal to enhance the OpenCL work group scheduling performance in a split driver model architecture for heterogeneous computational accelerator devices connected to the host through PCIe is a dynamic periodic polling-based parallel computing kernel task termination detection mechanism.

A. XDD Architecture and key features

As shown in Figure 3, the XPU Device Driver (XDD) is a platform device driver for executing OpenCL parallel computing applications and providing device initialization and management for the AB21 hardware device, which is a heterogeneous acceleration (XPU) SoC connected to the host system. XDD offers the XPU Driver Library (XDL) to support the XPU backend of the AB21 device within the SOCL development that was developed based on PoCL to ensure OpenCL 2.0 compatibility. The XPU device driver that was suggested has a model architecture of a split driver, which divides and processes tasks related to parallel computing kernel workgroup scheduling in a host OS and device OS at the same time. The XPU Platform Driver (XPD) that runs on the host system and the Inner-XPU Driver (IXD) that runs on

the AB21 device share and process scheduling metadata generation and workgroup scheduling via shared queues and scheduling metadata in device memory. The IXD driver comprises XEMIS Scheduler and Job Execution Checker components for workgroup scheduling. Using parallel computing tasks and scheduling metadata from XPD, the XEMIS Scheduler assigns work groups to available XEMIS on the XPU. The Job Execution Checker checks for parallel thread group termination on XEMIS by monitoring the execution status register through a dynamic period polling mechanism.

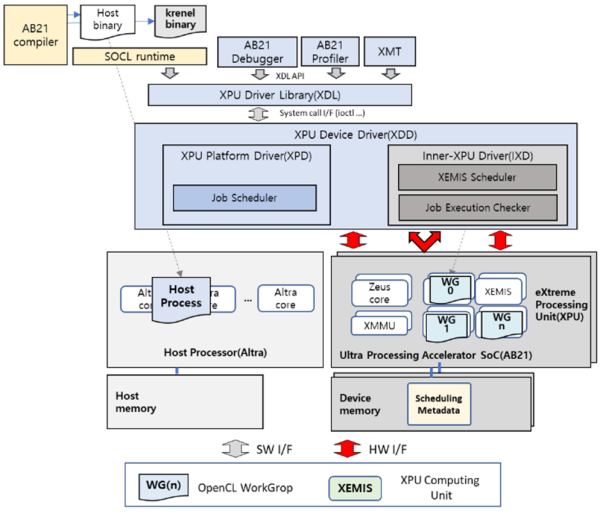

Fig. 3. Architecture and component diagram of the XDD software stack.

B. *Memory configuration and access control of the SoC device*

The AB21 device's physical memory is roughly divided into three regions: the device operating region, the shared metadata region, and the device allocation memory region. The first region is a memory space dedicated to the device. It contains the embedded operating system for device operation and management, drivers, and system software. The second memory area is a shared space between the host system and the device. It stores shared queues such as the Job queue and Command queue, as well as scheduling metadata like the Execution State of parallel application jobs used for XDD scheduling. This is a memory space commonly used by driver modules for scheduling and notifying commands, job requests, processing results, and errors of both the host and the device for XDD and IXD. When the XDD driver boots, it creates a shared queue that IXD initializes. This is done by sharing information that was set at boot time using AB21 registers. The memory data space of the device memory constitutes the third area, which is allocated and managed by the memory management mechanism of the IXD. The global memory space of the parallel operation kernel performs the memory copy of input/output data for the parallel computing application of the host system using DMA. The XMMU, integrated in the XPU, is responsible for the quick translation of addresses and access control of the memory allocated to the HW thread device. On the other hand, the XEMIS of the XPU manages the device memory.

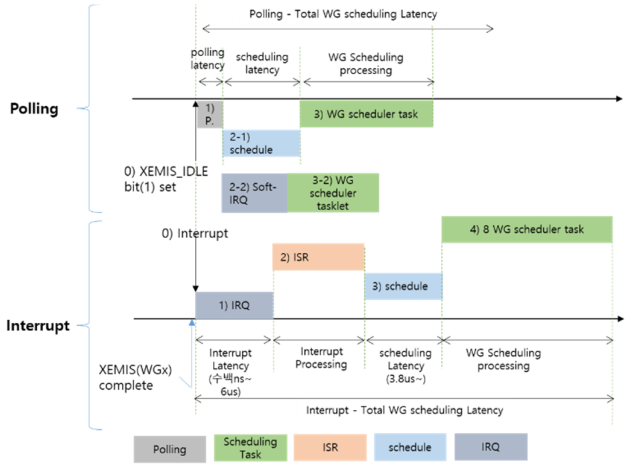

Fig. 4. Comparison of the cost of work-group scheduling on XPU: notification method of kernel task termination using polling versus interrupt.

C. *Work group scheduling with adaptive polling*

To support the concurrent execution of multi-user processes on a heterogeneous computational accelerator SoC with an integrated general-purpose core and computational accelerator core, a workgroup scheduling scheme is proposed. In an environment where no dedicated hardware scheduler exists within the computational accelerator SoC, OpenCL parallel computation kernel tasks are allocated efficiently to the parallel computation unit (XEMIS) and executed. IXD provides an execution termination detection mechanism through polling, based on the kernel execution status register of each XEMIS, using a dedicated general-purpose core. As shown in Figure 4, the method enables fast scheduling by rapidly detecting the execution termination of workgroups running in each XEMIS, compared to methods that use interrupts or fixed period polling. Moreover, by periodically calculating the average execution termination detection times and updating the polling cycle, the load on the SoC general-purpose core due to frequent polling is reduced. It offers a way to selectively apply execution result notification and scheduling methods through the AB21 device's scheduling control register.

IV. AB21 SoC Development Tools Support

We provide development tools, like management tools, debuggers, and profilers, to effectively debug, verify, and optimize different upper software stacks, including AB21 SoC device-based parallel applications, benchmarking tools, and system software. Extra APIs are available to support debugging and profiling functions in the XDD and XDL layers. The debugger and profiler requests are handled through the extension of shared queue commands and events. In addition to libraries that support SoCL Runtime, XDL also supports extra APIs, libraries, and driver functions for debugging, profiling, configuration, operation setup, and monitoring support.

A. *XMT (XPU Management Tool)*

XMT provides monitoring and configuration support for XPU operation similar to Nvidia's nvidia-smi tool for GPU accelerators. It focuses on XDD and its related configuration and status information. XDD provides essential information to XMT by using driver initialization information and XPU management knowledge from lower level components such as XPD and OS. Upon request, it transmits the allocated resource

information of the parallel computing kernel used by the active user processor in real-time.

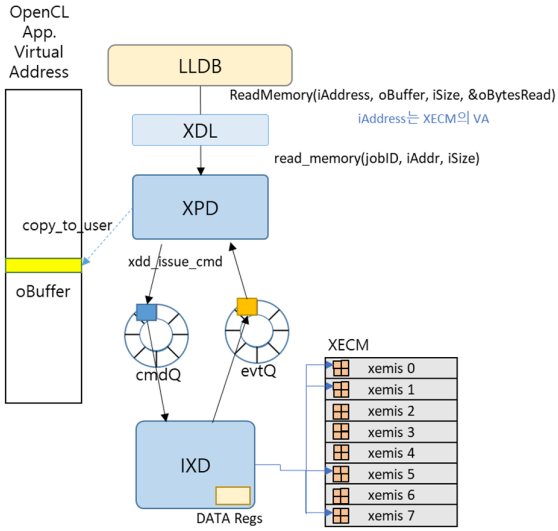

Fig. 5. An example of the conceptual operation diagram for XDD-based debugging functionality is the processing of memory information read requests from a running XECM thread.

B. AB21 Debugger support

The XDD and driver library, which serve as a lower backend of the LLDB debugger, have been expanded to support debugging of the AB21 SoC device. To achieve this, a debugging environment is established by utilizing the DM (Debug Module) of the calculation core (XECM) inside the AB21 SoC, a hardware accelerator that supports RISC-v ISA. IXD provides thread-level stepwise debugging for the parallel computation kernel present in the calculation core through DMI for DM (Debug Module) of XECM for debugging control and management processing requested through XPD's command queue. In addition to debugger support, XDL also offers 23 debug APIs covering 6 categories like kernel execution, memory access, read/write register, thread info, and breakpoint. As illustrated in Fig. 5, the command queue of XDD/IXD handles the request for processing the debugger function, which is then transferred to the DM of each XECM for processing. Upon execution of the command request and in case of occurrence of a SW or HW fault during debugging, the memory copy of the buffer area is performed through registers or DMA by making use of the event queue.

C. AB21 Profiler support

To support the AB21 accelerator device profiling tool, the XDL library and XDD can collect information on 16 key metrics registers, such as instruction and cache miss. XDL provides nine APIs, such as profiling start/stop and read profile data, while XDD reads the values of control registers and PMON registers inside the operation accelerator core (XEC). It then collects all metric data through DMA. The cost of collecting profiling data is minimized by sending profiling data in bulk to the user buffer area allocated to the host system.

V. Conclusions and Future Works

This paper presents the design and development of a split driver model device driver architecture with the aim of reducing parallel kernel scheduling overhead of XDD for the AB21 SoC, a heterogeneous accelerator SoC for the Supreme-K supercomputer prototype. We propose a dynamic polling-based termination detection mechanism with dynamic periods to enhance OpenCL workgroup scheduling performance for heterogeneous computational accelerator devices. Furthermore, we also modify device driver architecture and functions to create a DM-based accelerator kernel debugger and profiler, which is a computation core (XECM) inside a hardware accelerator supporting RISC-v ISA. This feature supports thread-level stepwise debugging for parallel computation kernel tasks performed in the accelerator. This feature handles debugger and profiler requests and reads registers using the shared, queue-based command queue and event queue of the XDD driver.

In the future, we plan to integrate and verify the AB21 SoC software stacks using OpenCL CTS in a multi-user, concurrent execution environment. We will perform various performance verifications and tests for the high-performance computing (HPC) sparse matrix application, using the in-house developed QAND application.

Acknowledgment

This research was supported by the Super Computer Development Leading Program of the National Research Foundation of Korea(NRF) funded by the Korean government (Ministry of Science and ICT(MSIT)) (No. 2021M3H6A1017683)

References

[1] 2022. TOP500 Lists. http://www.top500.org/lists/top500.

[2] Park, Y. M., et al. “The Innovation Engine for the Future: Supreme-K Supercomputer system”, in Proc. of the 2022 Summer Conference of The Institute of Electronics and Information Engineers (IEIE), 2022.

[3] Lyuh, C. G., et al. “XPU-based Ultra-High Performance Processor SoC Technology for Super Computer”, in Proc. of the 2022 Summer Conference of The Institute of Electronics and Information Engineers (IEIE), 2022.

[4] Lim, E. J., et al. “Design of Device Driver for Accelerator of Massively Parallel Processor for Supercomputer supporting OpenCL”, in Proc. of the 2022 Summer Conference of The Institute of Electronics and Information Engineers (IEIE), 2022.

[5] S. Mittal a nd J . S . Vetter, "A survey of CPU-GPU heterogeneous computing techniques", ACM Comput. Surv., vol. 47, no. 4, pp. 1-35, Jul. 2015