Operating System Support for Fine-grained Pipeline Parallelism on Heterogeneous Multicore Accelerators

Atsushi Koshiba
(Student and Presenter)
Tokyo University of Agriculture and Technology
koshiba@namikilab.tuat.ac.jp

Ryuichi Sakamoto
The University of Tokyo
r-sakamoto@hal.ipc.i.u-tokyo.ac.jp

Mitaro Namiki
Tokyo University of Agriculture and Technology
namiki@cc.tuat.ac.jp

On-chip special-purpose accelerators are a promising technique in the achievement of high-performance and energy-efficient computing. In particular, fine-grained pipelined execution with multicore accelerators is suitable for streaming applications such as JPEG decoders, which consist of a series of different tasks and process streaming data. CPUs that assign each task to appropriate accelerators and execute using pipeline parallelism achieve much performance gain.

Although accelerators have great potential of performance, the device driver overhead leads to performance degradation. In a pipelined execution, user processes such as OpenCL Runtime are responsible for executing tasks assigned to accelerators, controlling the direct memory access (DMA) for data transfers, and synchronizing all devices every pipeline stage. User processes are forced to communicate with the respective device drivers while accessing accelerators and DMAs and handling their interrupts. This user/kernel interaction causes a microsecond order overhead, which results in a performance penalty.

Some researchers propose an OS support for effective use of accelerators. [2] proposes an OS-level abstraction of GPUs and a programming model for streaming applications. However, its focus is limited to GPUs. [1] proposes an task scheduling scheme for efficient access to accelerators. However, this method focuses on fair sharing of accelerators during multi-tasking, and does not support pipelined execution.

To reduce the driver overhead, we propose an OS support mechanism to eliminate interactions between user-mode applications and kernel-mode drivers. We present a kernel module named *Accelerator Pipelining Controller (APC)*, which is responsible for managing all accelerators and DMAs. The APC analyzes the accelerator usage patterns of pipelining applications and manages all task executions and data transfers until all pipeline stages are complete. Our approach supports applications that are written in a producer-consumer model using OpenCL APIs.

The APC uses *a pipelining table*, which represents the executing tasks on devices (accelerators, DMAs) of each pipeline stage, to execute all the pipeline stages without invoking the associated user processes. The table is automatically generated by profiling the OpenCL application in advance. The profiler detects the task dependency and data allocation by analyzing the OpenCL kernels and their data access patterns. Next, it creates the table for the application. The APC reads the pipelining table of the target application at the beginning of the execution. It, then, controls the accelerators and DMAs according to the table.

To estimate the effectiveness of our method, we developed a prototype of heterogeneous multicore platform, which consists of a host processor (ARM Cortex-A9) and an image processing accelerator. We also implemented the device driver for the accelerator on Linux 4.4.0. Then, we executed programs on the accelerator using the driver, and measured the execution time in one pipeline stage. The result shows that the consequent overhead occupies more than 50% of the execution time in one stage. Because our method gets rid of the interactions between user processes and drivers, we expect that our method can improve processing speed by up to 1.8X. We also expect that our method can be more effective as the number of accelerators increases.

References

[1] K. Menychtas, K. Shen, and M. L. Scott. Disengaged scheduling for fair, protected access to fast computational accelerators. *SIGPLAN Not.*, 49(4):301–316, Feb. 2014. .

[2] C. J. Rossbach, J. Currey, M. Silberstein, B. Ray, and E. Witchel. Ptask: Operating system abstractions to manage gpus as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 233–248. ACM, 2011.

CONF'yy, Month d–d, 20yy, City, ST, Country.
Copyright © 20yy ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00.
http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

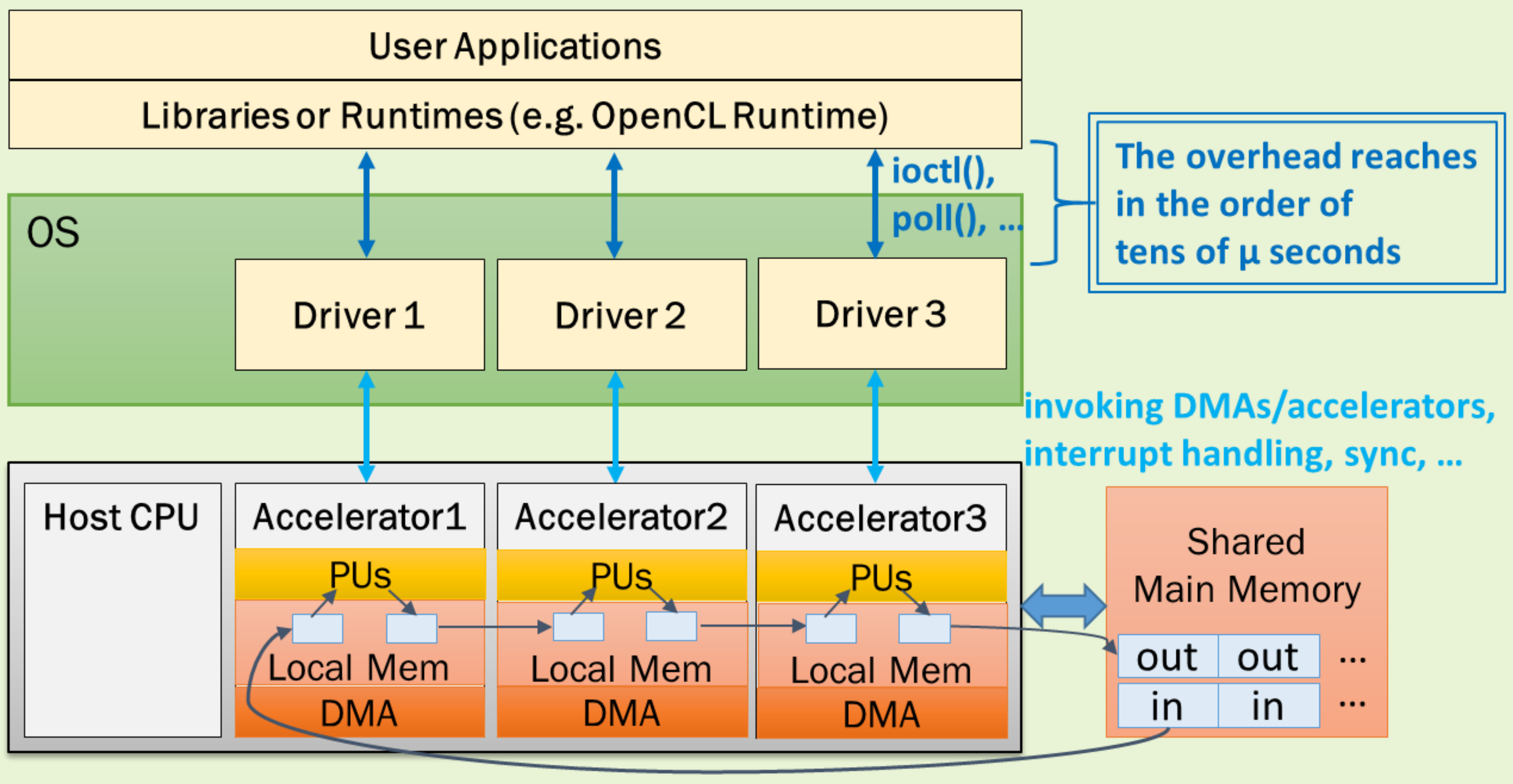
Operating System Support for Fine-grained Pipeline Parallelism on Heterogeneous Multicore Accelerators

Atsushi Koshiba (Student)†1, Ryuichi Sakamoto†2, Mitaro Namiki†1
†1Tokyo University of Agriculture and Technology, †2The University of Tokyo

Background

- **Fine-grained pipeline parallelism (FGPP) on accelerators achieves high-performance and energy-efficient computing** [1]
  - ✓ FGPP is a technique to overlap task executions and data transfers
    - Assigning each task to appropriate accelerators
    - Pipelining data between the accelerators
  - ✓ Effective for streaming apps (e.g., JPEG encoder/decoder)
- **User-level control of FGPP leads to poor performance**
  - ✓ User-level apps (e.g., OpenCL Runtime) have to use device drivers to execute tasks on accelerators, control DMAs, and synchronize them
    → causing frequent context switches
  - ✓ The switching overhead can occupy roughly 50% of the execution time (shown in our evaluation results)

Traditional Approach



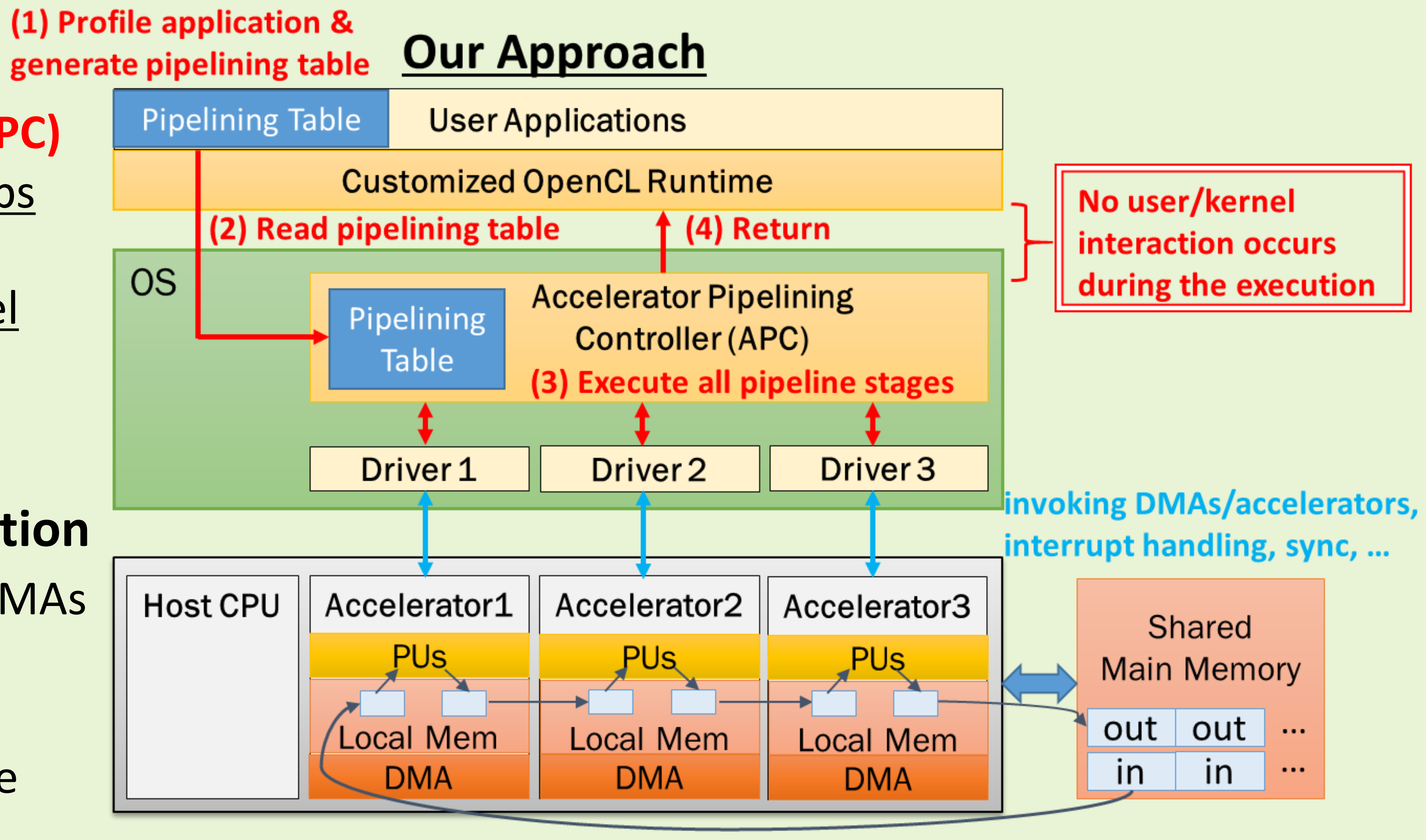
The overhead reaches in the order of tens of μ seconds

invoking DMAs/accelerators, interrupt handling, sync, …

Proposed Ideas

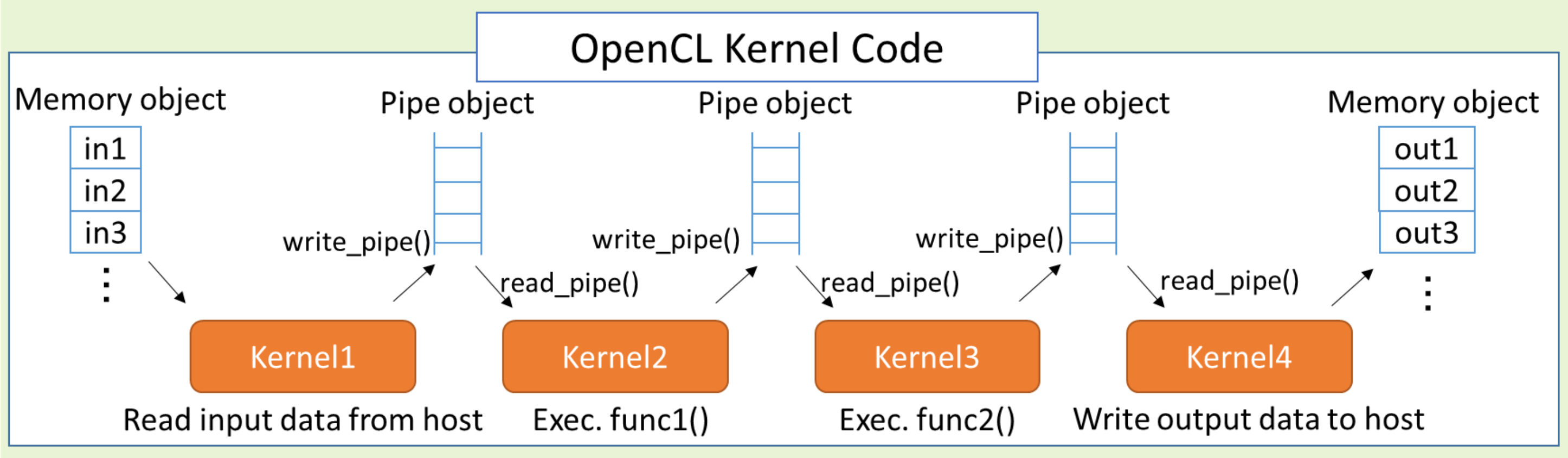
- **OS-level run-time support for FGPP on accelerators**
  - ✓ A new kernel module named **Accelerator Pipelining Controller (APC)**
  - ✓ APC manages all accelerators and DMAs without invoking user apps
    - reduce the overhead related to switching user/kernel mode
  - ✓ Applicable to OpenCL apps. written in a producer-consumer model
  - ✓ Supports accelerators which consist of basic functions: processing units (e.g., ALUs), a local memory and a DMA
- **Code offloading based on profiled task-dependency information**
  - ✓ **The pipelining table** represents executing tasks on accelerators/DMAs and input/output data allocation in pipeline stages
  - ✓ The table is automatically generated by profiling (shown below)
  - ✓ APC controls accelerators/DMAs according to the table at run-time
    → causes NO user/kernel interaction while executing the app

Our Approach



No user/kernel interaction occurs during the execution

invoking DMAs/accelerators, interrupt handling, sync, …



Analyze task dependency
&
Generate pipelining table

	Stage1	Stage2	Stage3	Stage4	Stage5
DMA1	src. : host dest. : accel.1 size : 0x800	src. : host dest. : accel.1 size : 0x800	src. : host dest. : accel.1 size : 0x800	src. : host dest. : accel.1 size : 0x800	src. : host dest. : accel.1 size : 0x800
Accelerator1		exec func1()	exec func1()	exec func1()	exec func1()
DMA2			src, dest, size	src. dest, size	src. dest, size
Accelerator2				exec func2()	exec func2()
DMA3					src. dest, size

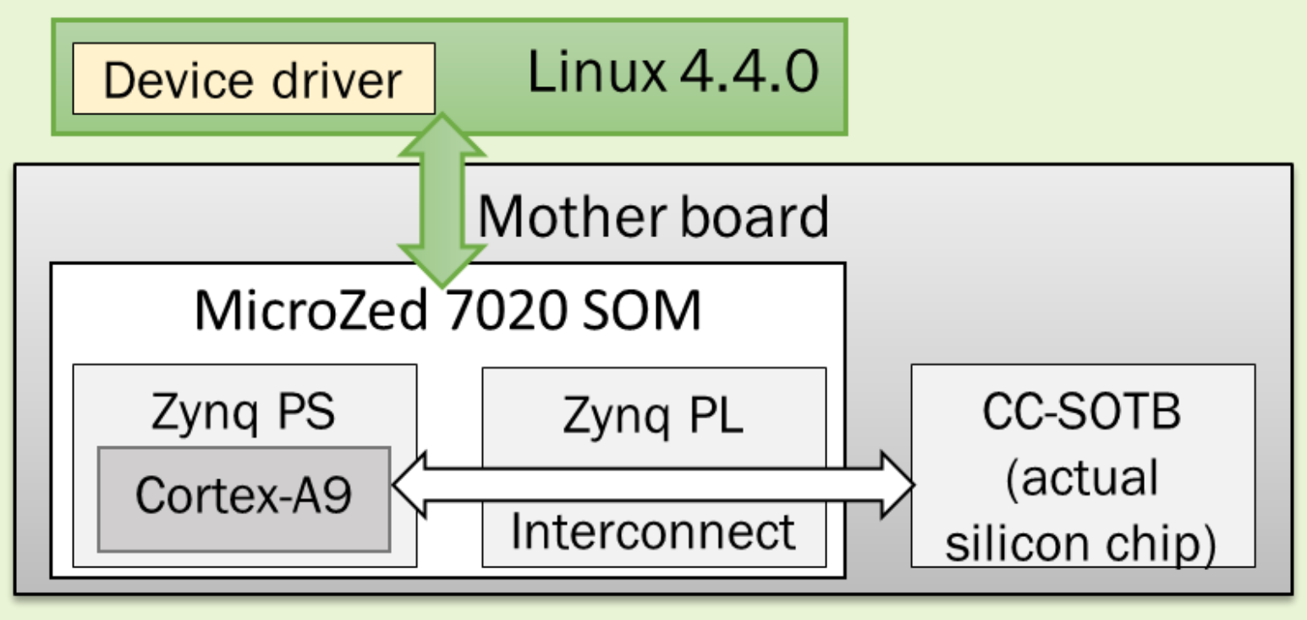
…

Describe tasks executed on each device every pipeline stage

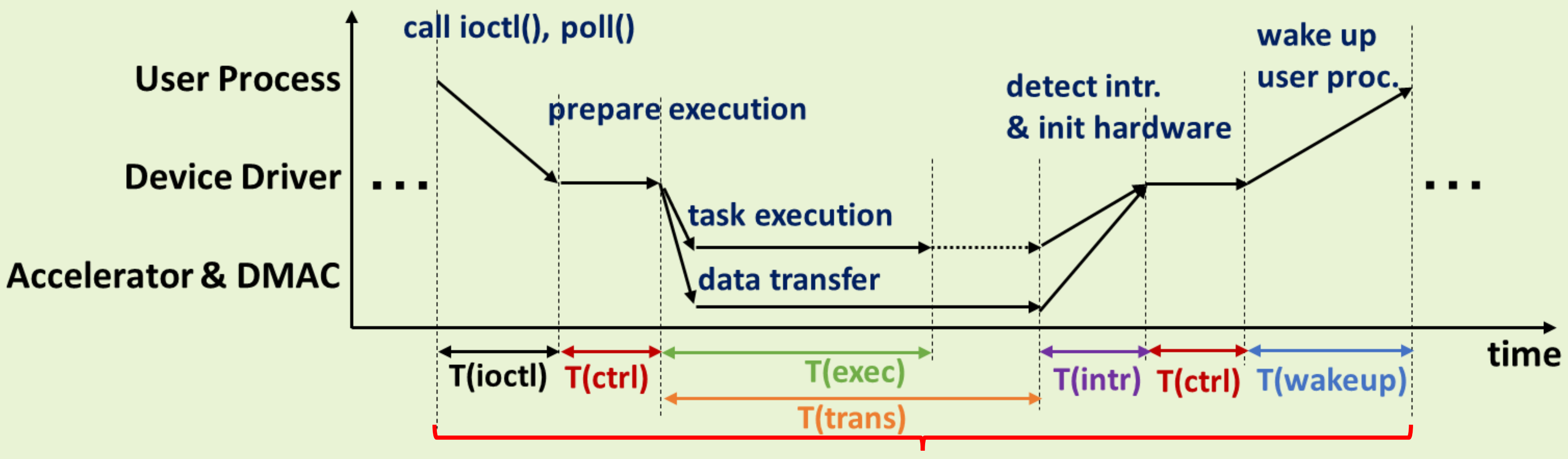
Evaluation

- **Implementation of our evaluation environment**
  - ✓ Implemented a prototype of a heterogeneous multi-core platform

Item	Specification
Main platform	MicroZed 7020 SOM
Host OS	Linux 4.4.0
Host CPU	ARM Cortex-A9 @ 667MHz (1GB RAM)
Accelerator	CC-SOTB [1] @ 60MHz (4kB local memory)

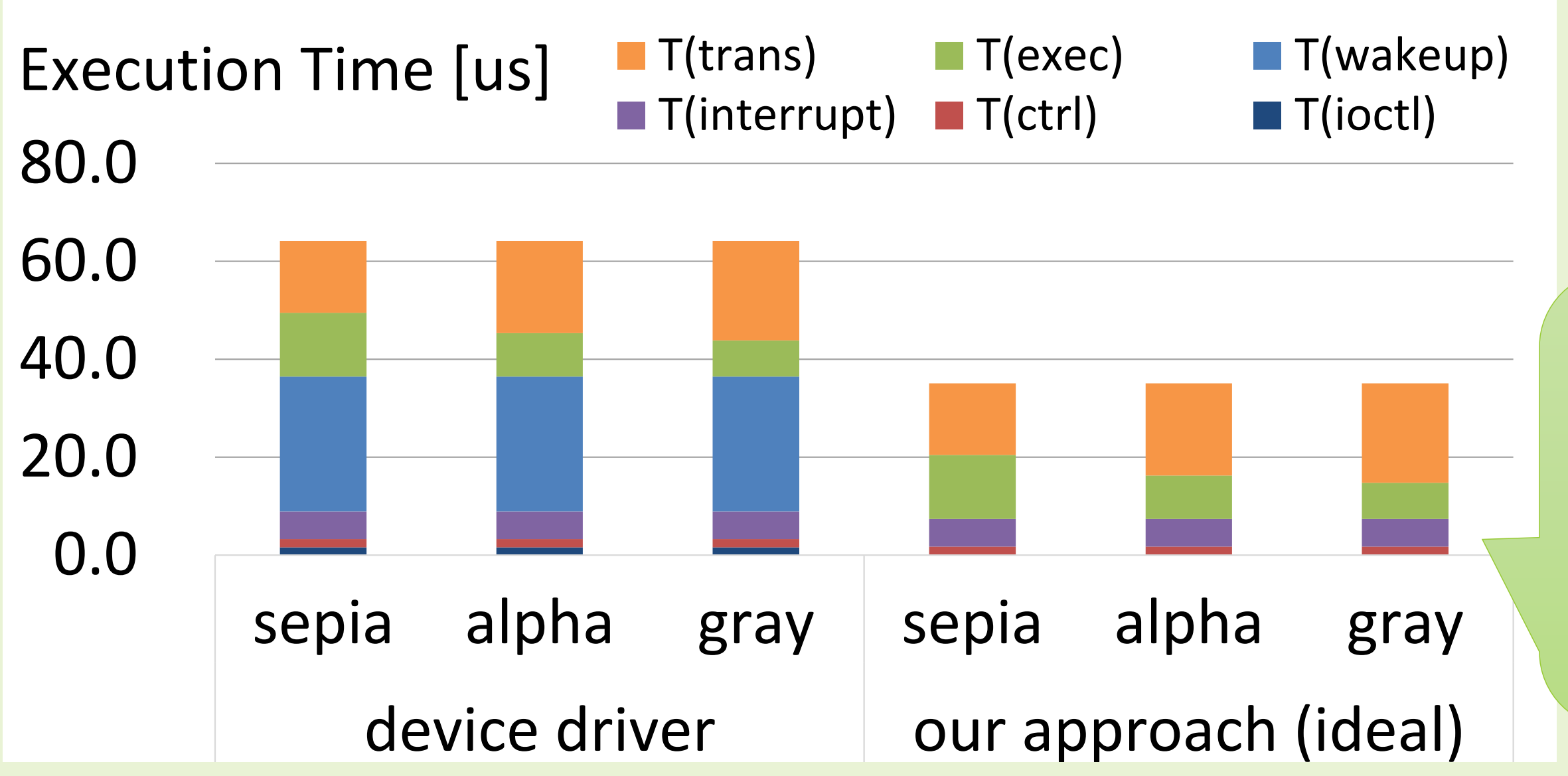


- **Analyzing device driver overhead**
  - ✓ Evaluated the execution time of image processing apps in one pipeline stage
  - ✓ Measured T(ioctl), T(ctrl), … independently using hardware counters



Execution time in one pipeline stage

- **Results**
  - ✓ **Our approach improves processing speed by up to 1.8x in an ideal condition**
    - Used one accelerator and DMAs
    - Evaluated a sepia filter, an alpha blender, and a gray filter
    - Input data size processed in one stage was 1.5 kB
  - ✓ Our approach can be more effective as the number of accelerators increases



Since our approach causes no User/OS interaction, T(ioctl) and T(wakeup) are eliminated

※ We estimated the results of "our approach" based on the results of "device driver"

- **Future Work**
  - ✓ Implement a prototype of the APC and the OpenCL code profiler
  - ✓ Implement a heterogeneous platform with multi-core accelerators
  - ✓ Evaluate streaming applications using multiple accelerators
  - ✓ Take interference in multi-tasking into account

[1] Nasibeh Teimouri et al., "Revisiting accelerator-rich CMPs: challenges and solutions", DAC 2015.
[2] Koichiro Masuyama et al., "A 297MOPS/0.4mW Ultra Low Power Coarse-grained Reconfigurable Accelerator CMA-SOTB-2," ReConFig 2015.

Acknowledgment:
This work was supported by Grant-in-Aid for JSPS Research Fellow 16J06711.