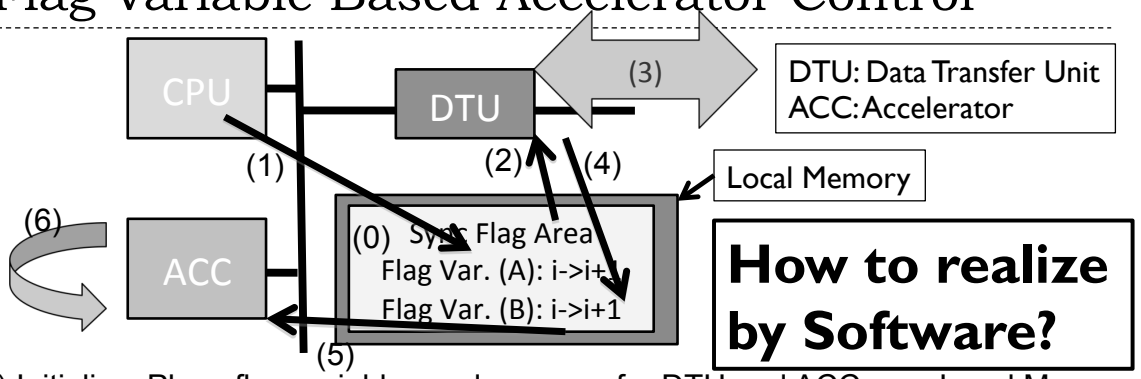
OSCAR API v2.1 with Flexible Accelerator Control Facilities

Keiji Kimura, Waseda University

Heterogeneous Computing: Current Connection between CPU and Accelerators

- Heterogeneous Computing is widely used:
  - Required Performance by Target Applications
  - Power-efficiency of Accelerators
- Connection between CPU and Accelerators
  - Tightly-coupled Accelerators
    - Extending Instruction Set
  - Attach via Bus (cf. GPU)
    - Large Control and Data Transfer Overhead
    - CPU and Accelerators should not communicate each other.
      - Flexible Accelerator Control is DIFFICULT
- New Way of Connection is Required

CPU, Accelerator and DTU: Flag Variable Based Accelerator Control



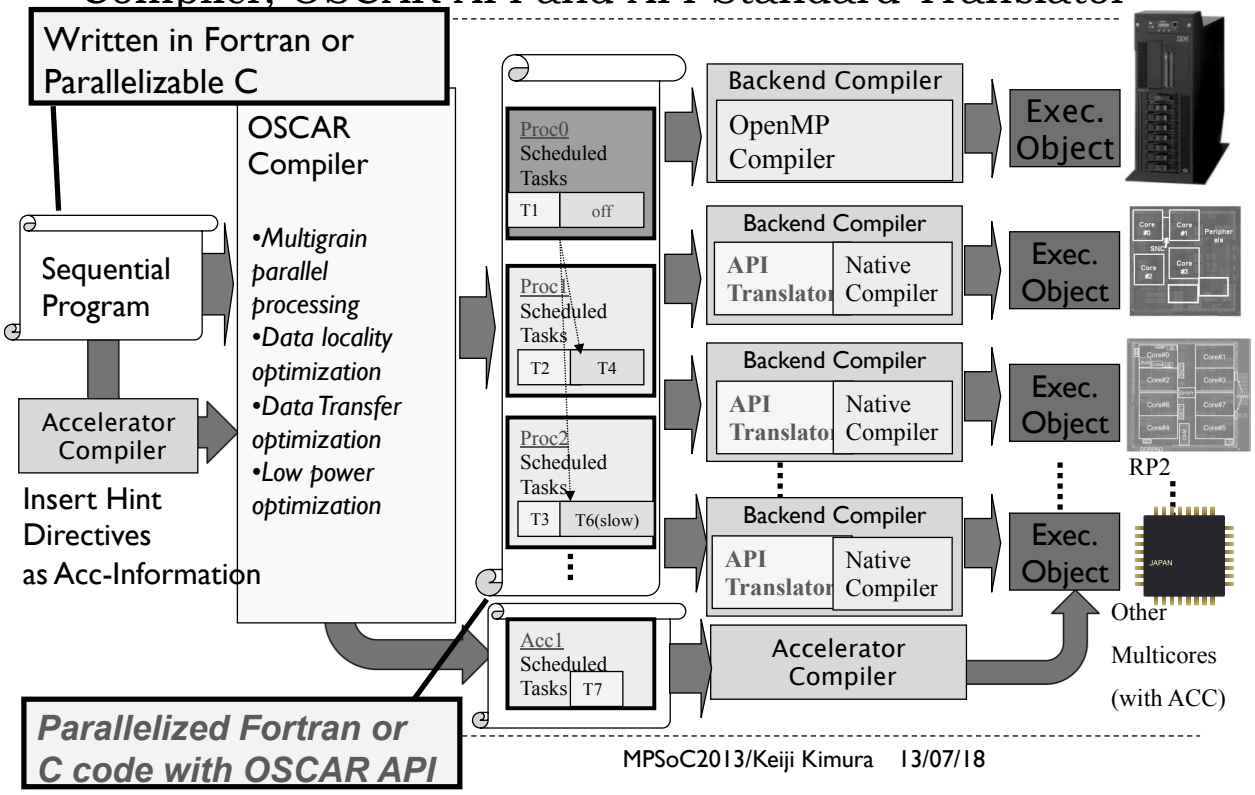
(0) Initialize: Place flag variables and program for DTU and ACC on a Local Memory
(1) CPU increments Flag Variable-A.
(2) DTU checks whether Flag Variable-A is incremented or not.
(3) DTU starts data transfer after detecting Flag Variable-A's increment.
(4) DTU increments Flag Variable-B after data transfer.
(5) ACC checks whether Flag Variable-B is incremented or not.
(6) ACC starts its execution after detecting Flag Variable-B's increment.

CPU, DTU and ACC can be executed simultaneously.
The execution timing of them can be notified by Flag Variables

Overview of OSCAR API v2.0 (before 2.1)

- Targeting mainly real-time consumer electronics devices
  - Embedded computing
  - Various kinds of memory architecture
    - SMP, local memory, distributed shared memory, non-coherent cache ...
  - Power control mechanisms
  - Accelerators
- Based on the subset of OpenMP
  - Very popular parallel processing API
  - Shared memory programming model
  - Supporting both of C and Fortran
- Eight Categories
  - Parallel Execution
  - Memory Mapping
  - Data Transfer
  - Power Control
  - Timer
  - Synchronization
  - Accelerator
  - Cache Control

Application Development Environment with OSCAR Compiler, OSCAR API and API Standard Translator



List of Directives of OSCAR API v2.0 (22 directives)

- Parallel Execution API
  - parallel sections (*)
  - flush (*)
  - critical (*)
  - execution
- Memoay Mapping API
  - threadprivate (*)
  - distributedshared
  - onchipshared
- Synchronization API
  - groupbarrier
- Data Transfer API
  - dma_transfer
  - dma_contiguous_parameter
  - dma_stride_parameter
  - dma_flag_check
  - dma_flag_send
- Power Control API
  - fvcontrol
  - get_fvstatus
- Timer API
  - get_current_time
- Accelerator
  - accelerator_task_entry
- Cache Control
  - cache_writeback
  - cache_selfinvalidate
  - complete_memop
  - noncacheable
  - aligncache

2 hint directives for OSCAR compiler
- accelerator_task
- oscar_comment

(* from OpenMP)

Newly Added Directive to OSCAR API v2.1

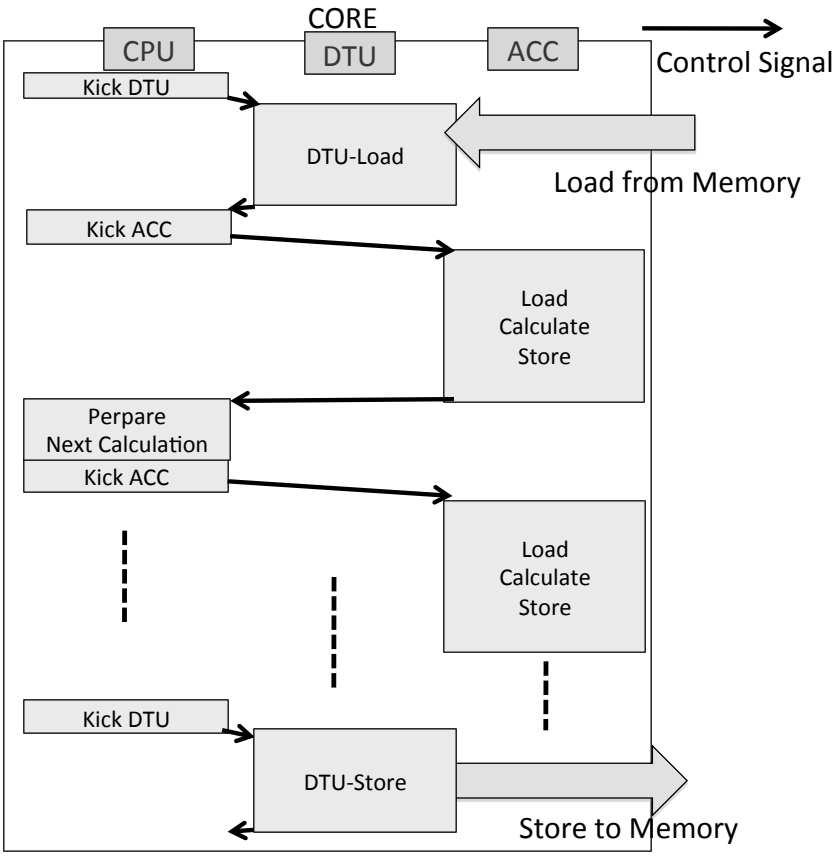
- accelerator_task_entry_nonblocking
  - Specify the entry function to be executed on accelerators
  - Execute the specified functions in non-blocking manner
- acc_flag_send
  - Send synchronization flag from an accelerator
- acc_flag_check
  - Check synchronization flag at an accelerator
- ex)
  - #pragma oscar accelerator_task_entry_nonblocking oscartask_loop2
    The function "oscartask_loop2" will be executed on accelerators.
  - #pragma oscar acc_flag_send(flag1, ver1)
    assign ver1 to flag1 for synchronization
  - #pragma oscar acc_flag_check(flag2, ver2)
    check whether the value of flag2 becomes same as ver2 or not

Very Simple Extension

Sample

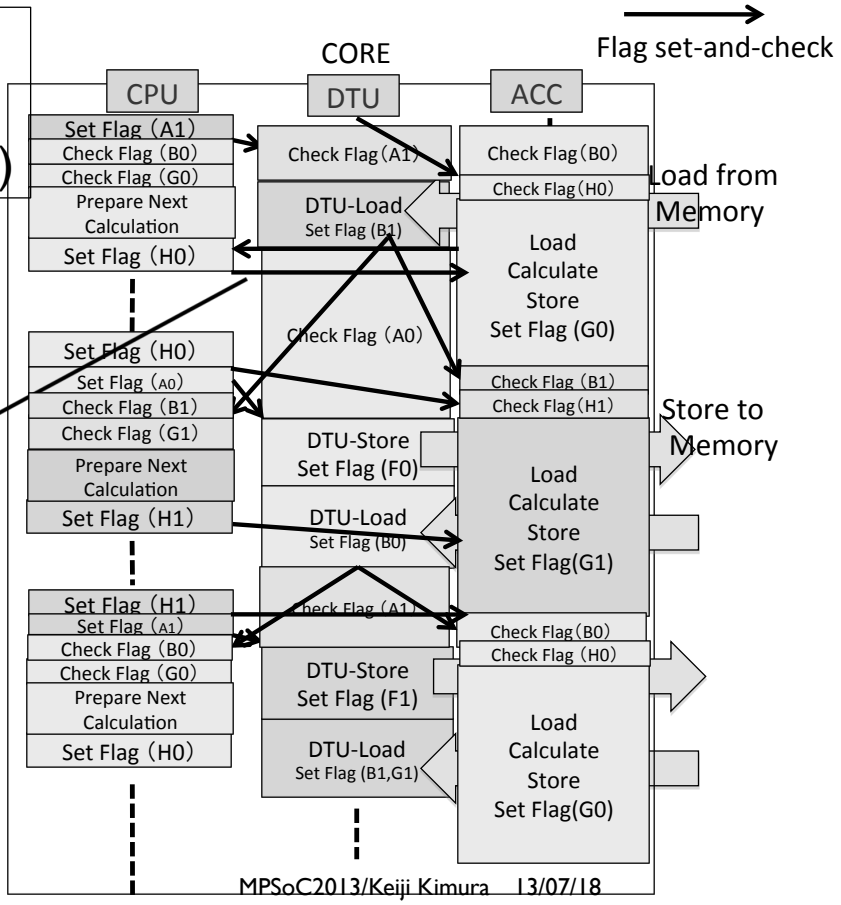
```
/* file: sample.VC2.c */
extern int flag1, flag2;
#pragma oscar distributedshared vpc(2) (flag1, flag2)
extern int y[10];

#pragma oscar accelerator_task_entry controller(2) \
    oscartask_CTRL2_loop1
#pragma oscar accelerator_task_nonblocking oscartask_loop2

void oscartask_CTRL2_loop1(int *x)
{
    int i;
    for (i=0; i < 10; i++)
        x[i]++;
}

void oscartask_loop2()
{
    int i;
#pragma oscar_acc_flag_check(flag0, 1)
    while (flag0 != 1);
    for (i = 0; i < 10; i++)
        y[i]++;
#pragma oscar_acc_flag_send(flag1, 2)
    flag1 = 2;
}
```

Example Execution Image (traditional)



Example Execution Image (OSCAR API v2.1)

The execution timing of CPU, DTU and ACC are controlled by Flag Variables.



Summary

- OSCAR API v2.1
  - One directive added for flexible accelerator control
- Flag Based CPU, DTU and Accelerator Control
  - They execute there own program simultaneously.
  - They communicate via Flag Variables each other.
    - Overlap Execution Realizes High Execution Efficiency.
      - Hiding data transfer and control overhead
- The Specification of OSCAR API can be downloaded from our web page:
  - http://www.kasahara.cs.waseda.ac.jp/