A Design of Hybrid Operating System for a Parallel Computer with Multi-Core and Many-Core Processors

Mikiko Sato[1,5] Go Fukazawa[1] Kiyohiko Nagamine[1] Ryuichi Sakamoto[1]
Mitaro Namiki[1,5] Kazumi Yoshinaga[2,5] Yuichi Tsujita[2,5] Atsushi Hori[3,5]
Yutaka Ishikawa[3,4]

[1]Tokyo University of Agriculture and Technology [2]Kinki University
[3]RIKEN Advanced Institute for Computational Science
[4]University of Tokyo [5]Japan Science and Technology Agency, CREST

Outline

- Background
- Motivation
- Design of Multi-core & Many-core system
  - Hardware architecture
  - Software architecture
- Prototype system and evaluation
- Conclusion

Background of this study

- Supercomputers are predicted to achieve exaflop performance by 2018.

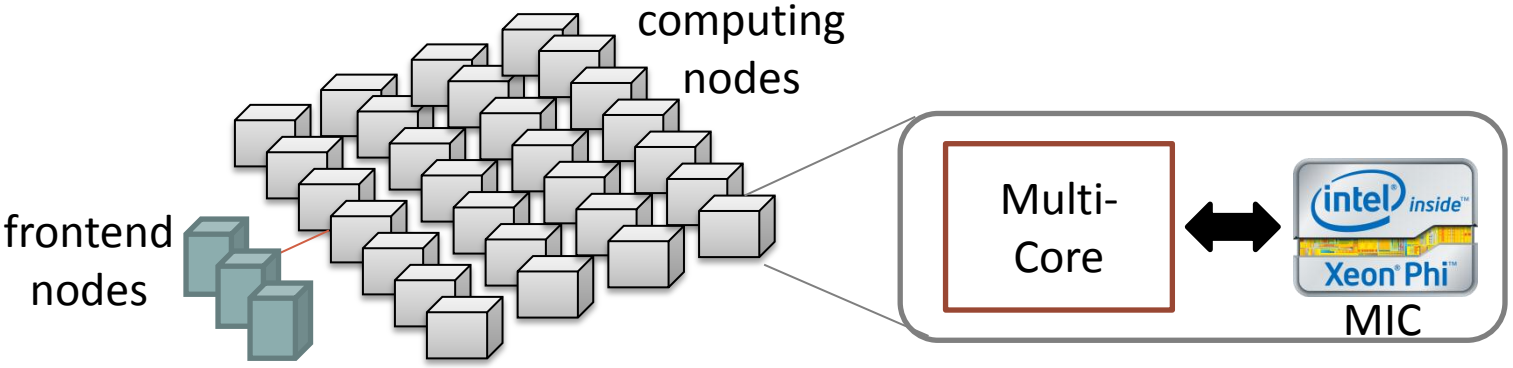
What kind of the system architecture can achieve it?

- Node architecture of the supercomputer
  - Multi-core processor nodes
    - Sequoia, Mira, Fermi, JuQUEEN (PowerBQC 16C)
    - K computer (SPARC64 VIIIfx)
    - SuperMUC (Xeon E5-2680 8C)
    - Jaguar (Opteron 6274 16C)
  - Multi-core and GPGPU nodes
    - Tianhe-1A (Xeon+NVIDIA 2050)
    - Nebulae (Xeon+NVIDIA 2050)
    - TSUBAME 2.0 (Xeon, NVIDIA 2050)

Our target

- Multi-core and Many-core processor combination
  - Intel Many Integrated Core (MIC)
    - *Xeon Phi* (Knights Corner)
    - multiple simple X86 cores (> 50 cores)

  → Plan to apply MIC to *the computing node* to achieve highly parallel processing!

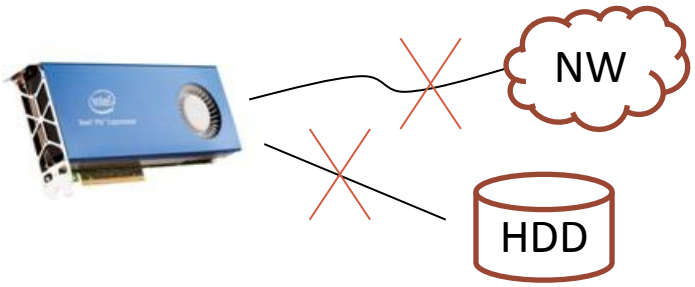


Motivation and Approach

- The reason of choosing MIC
  - ➢ Compatibility with x86 programming models

  → the *System software* has to support APIs on *MIC*
    - UNIX/POSIX-API (memory management, I/O access etc..)
    - Parallel programming API (MPI/Thread etc..)
  - ➢ MIC is possible to run OS

MIC has few memories per core (8GB RAM are used by >50cores)



I/O cannot be connected directly

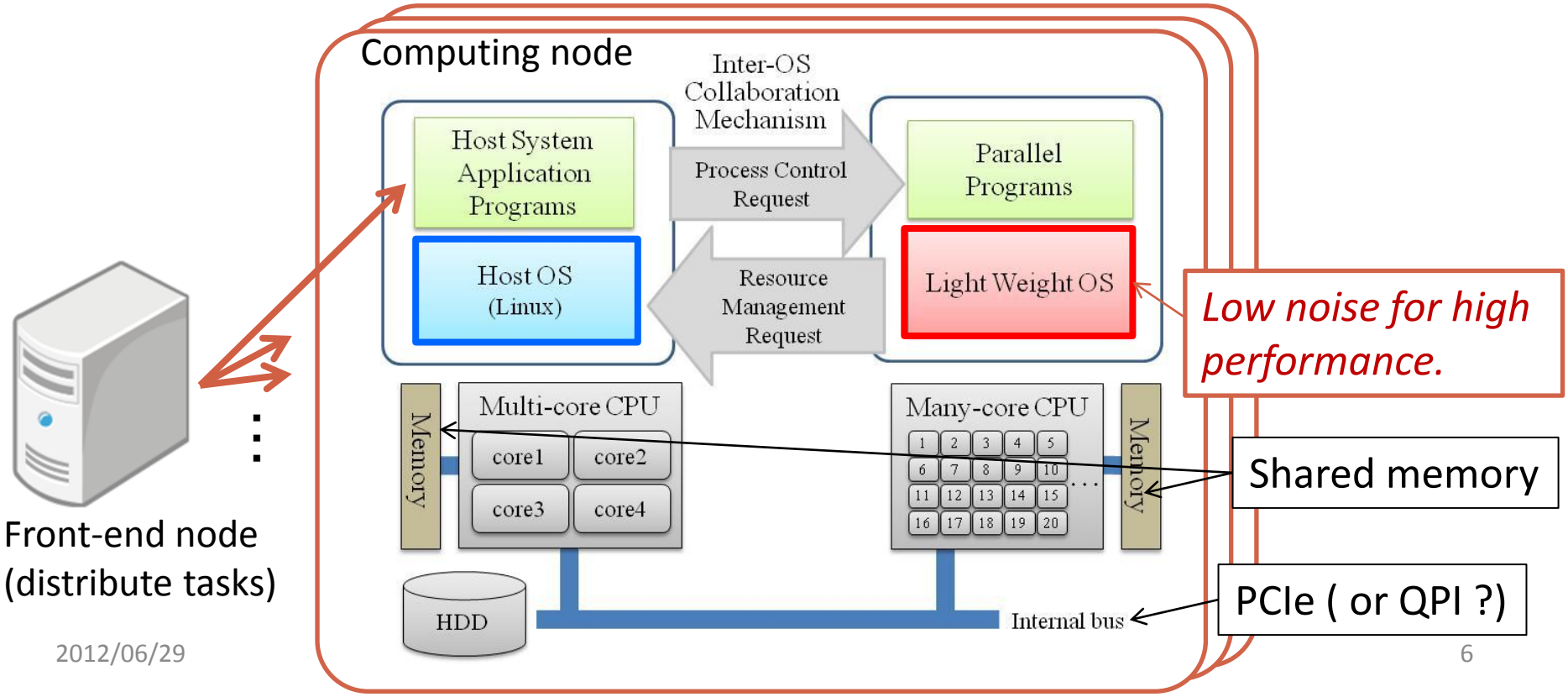
→ The Operating System on MIC should be light weight!
  - Host OS performs some functions instead of light weight OS.

What kind of Host OS supports are effective for LWOS?
(process, memory, and I/O management)

Hybrid computer system overview

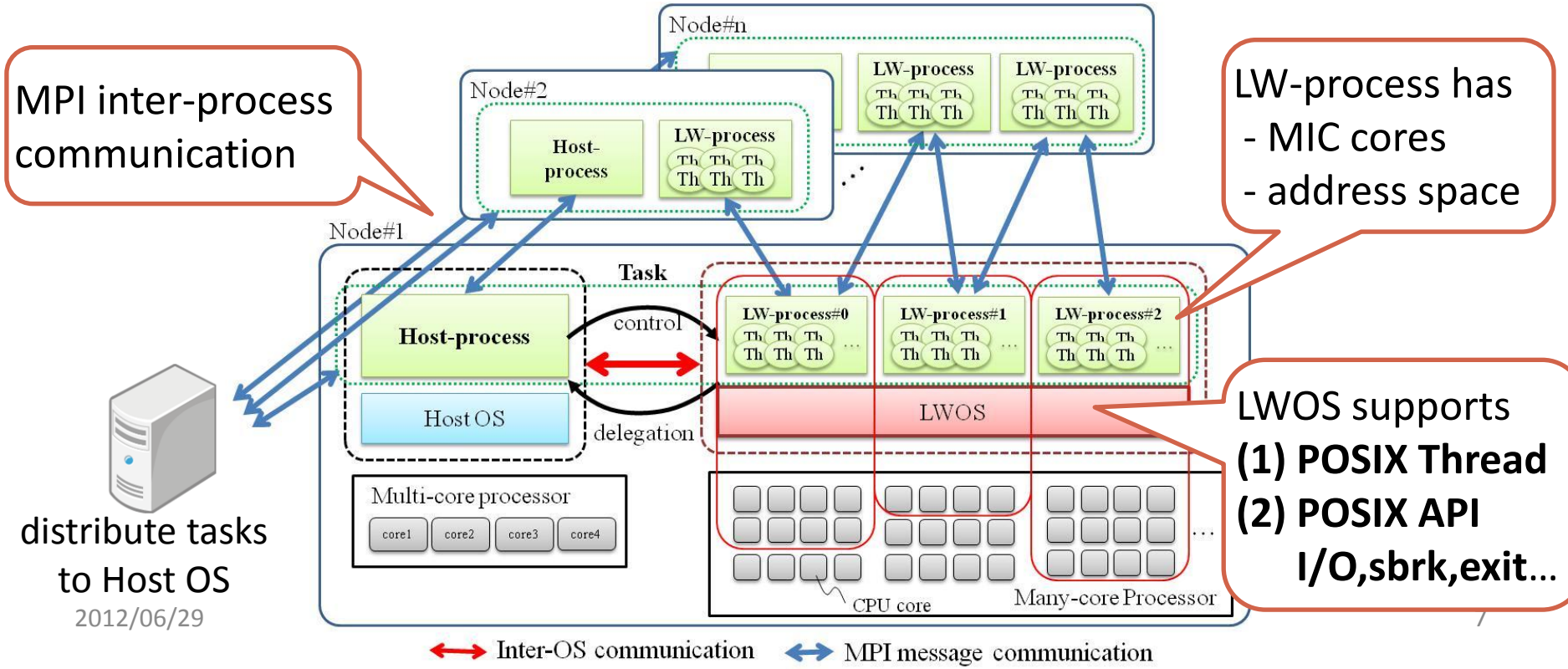
- Linux on Multi-core CPU works as Host OS
  - I/O devices and Many-core resources management
- Light Weight OS on Many-core CPU (MIC)
  - Thread management with low noise

similar to recent LWK (CNK, Kitten, Catamount)



Process model

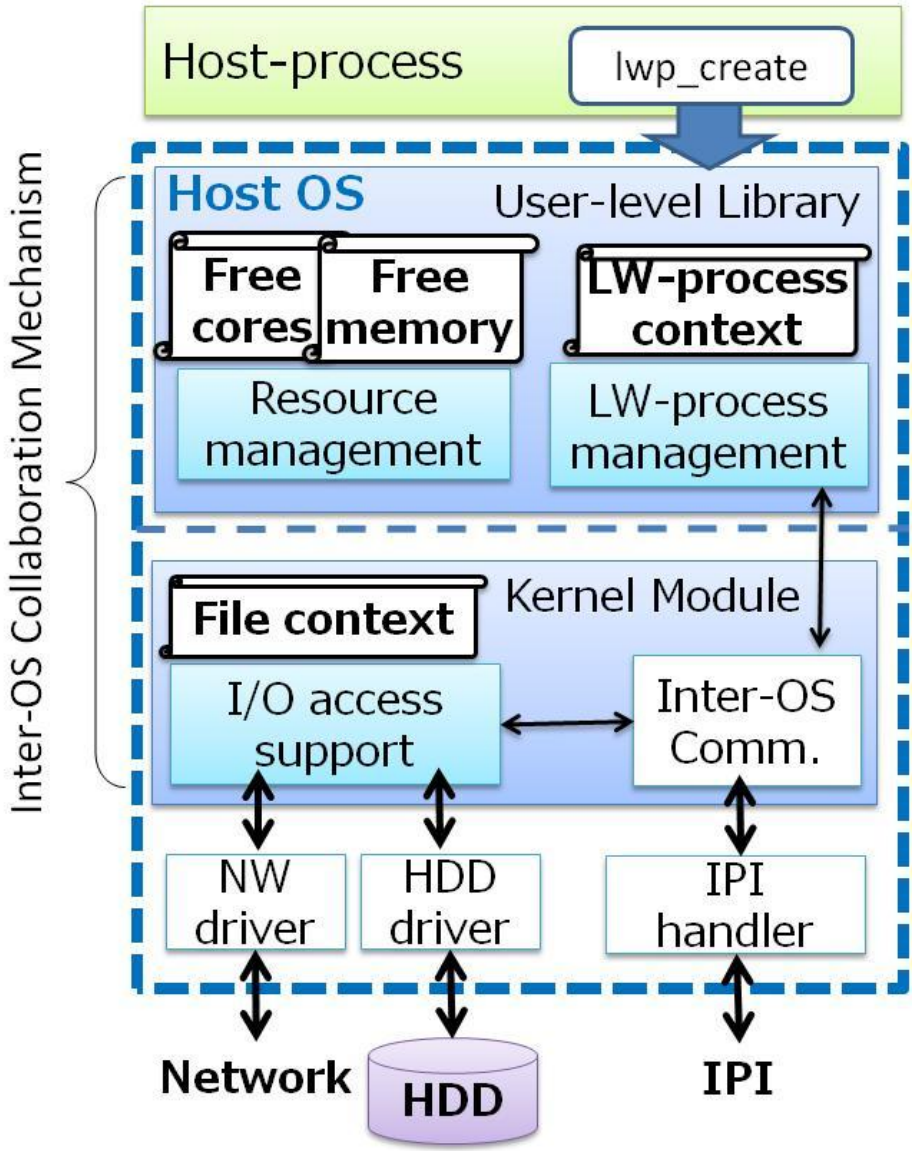
- Host-process and LW-processes are formed into a group (*Task)* .
  - *Host-process* controls *LW-process* (create, delete, etc..)
  - ***POSIX Threads*** execute using the *LW-process* resources.
  - MPI is used for inter-process communication
    - The main node distributes the Task group to each node using MPI



LW-process Management

- Host OS roles
  - (1) The Host OS manages the cores of the Many-core CPU and assigns free cores to the LW-processes
  - (2) The Host OS manages the physical memory of the many-core processor and continuously assigns memory areas to LW-processes.
  - (3) The Host OS manages I/O access requests from Threads on LWOS.
- LWOS role
  - ➢ Thread management
    - LWOS delegates the resource management function to the Host OS.

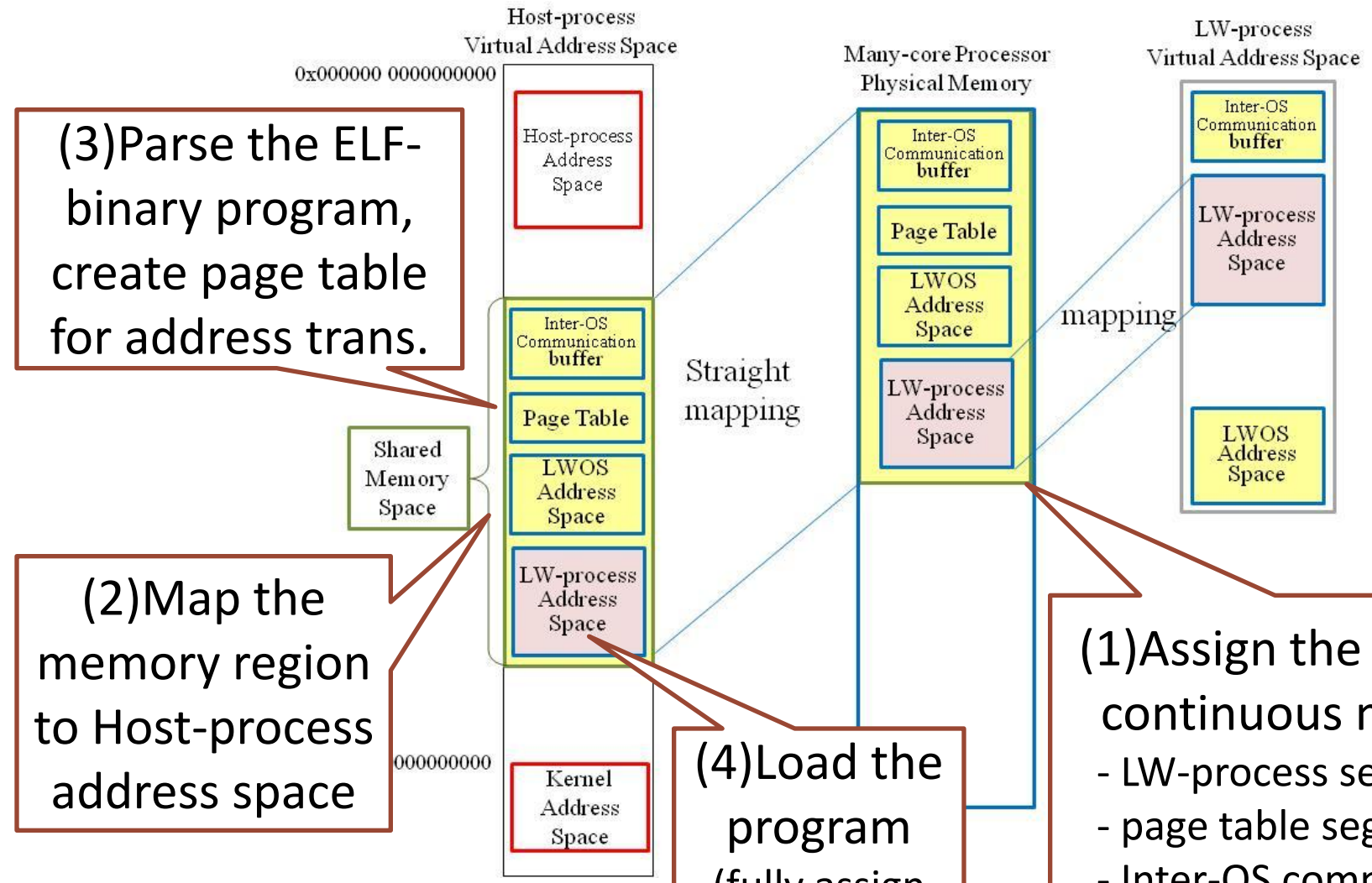
Host OS structure (user-level)



- LW-process management
  - ➢ control the LW-process

  (Create, Destroy, Suspend, Resume)
  - load the ELF-binary program
  - complete the page table
  - ➢ keep the LW-process context
  - Physical core on Many-core
  - Physical memory of Many-core
  - The start address of Page Table
  - Inter-OS comm. buffer address
- Resource management
  - ➢ assign physical resources
  - ➢ keep the free resources

Memory mapping for LW-process



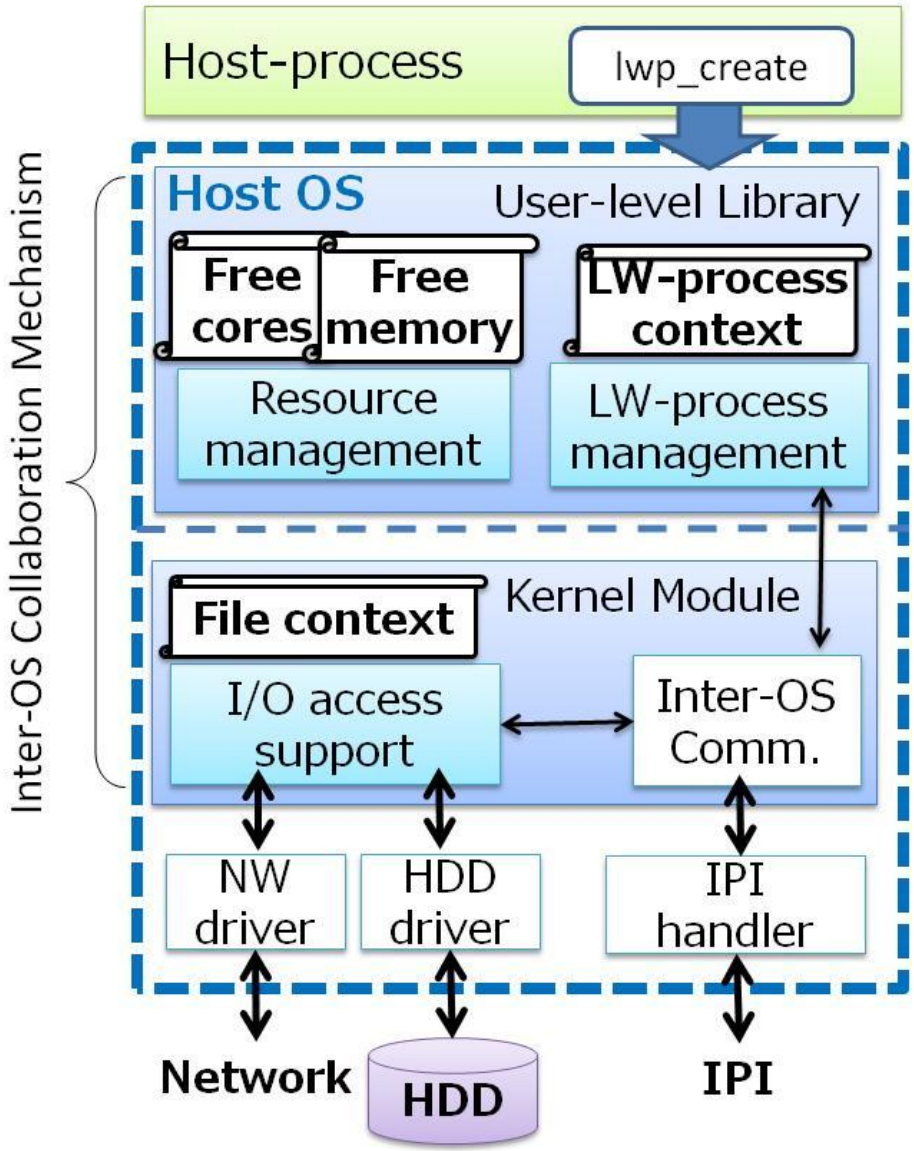
(3)Parse the ELF-binary program, create page table for address trans.

(2)Map the memory region to Host-process address space

(4)Load the program (fully assign stack & heap)

(1)Assign the required continuous memory
- LW-process segment
- page table segment
- Inter-OS comm. buffer

Host OS structure (kernel-level)

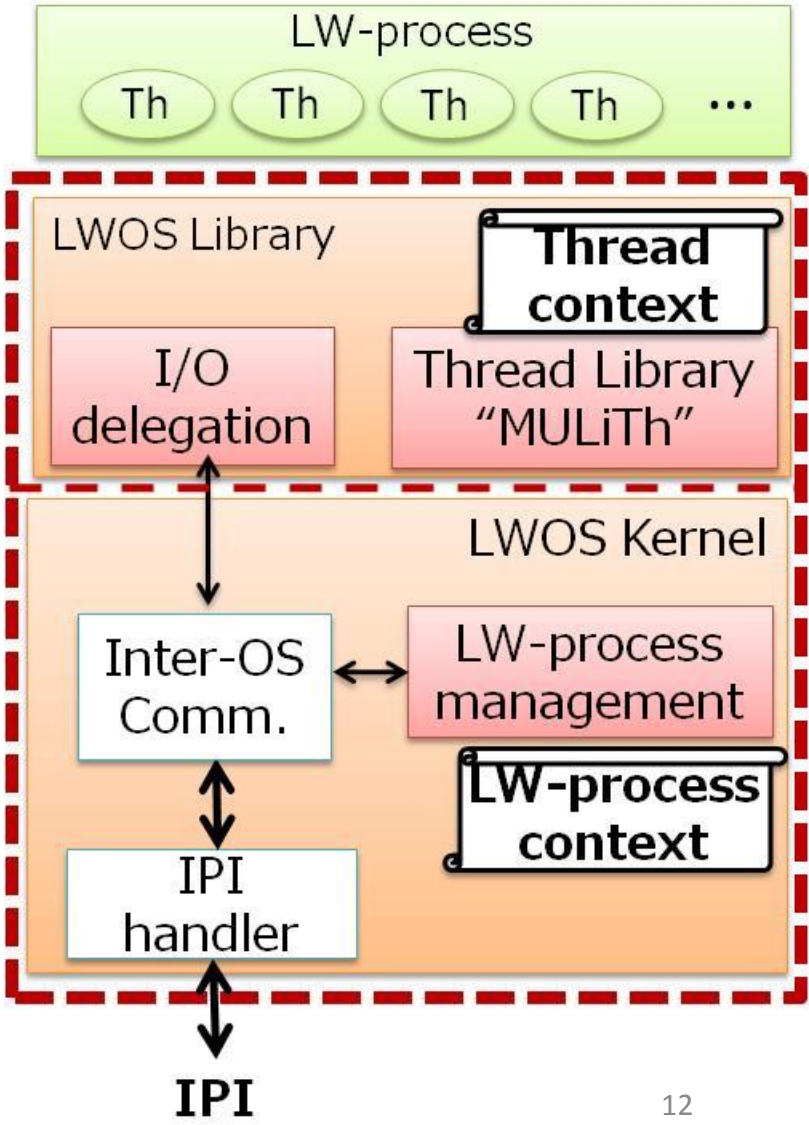


I/O access support

- Receive I/O access requests
  (open, close, read, write, etc..)
  - Buffer address
    (LW-process virtual address)
  - File descriptor
  - Access size
- I/O access at Kernel module
  - change into virtual address on Host OS
  - Access through Linux file system
- Keep the file context
  - The information of opened file
  - Check it when accessing the file.

LWOS structure (User Level)

- POSIX Thread Library
  - Each core on Many-core CPU is virtualized to a thread.
  - Thread management
    - pthread_create, join, etc..
    - Non-preemptive execution
    - Keep the thread contexts
- POSIX API Delegation Library
  - Set the arguments of POSIX API to Inter-OS communication buffer
  - Notice the delegation to Host OS
    - via Kernel level **IPI handler**

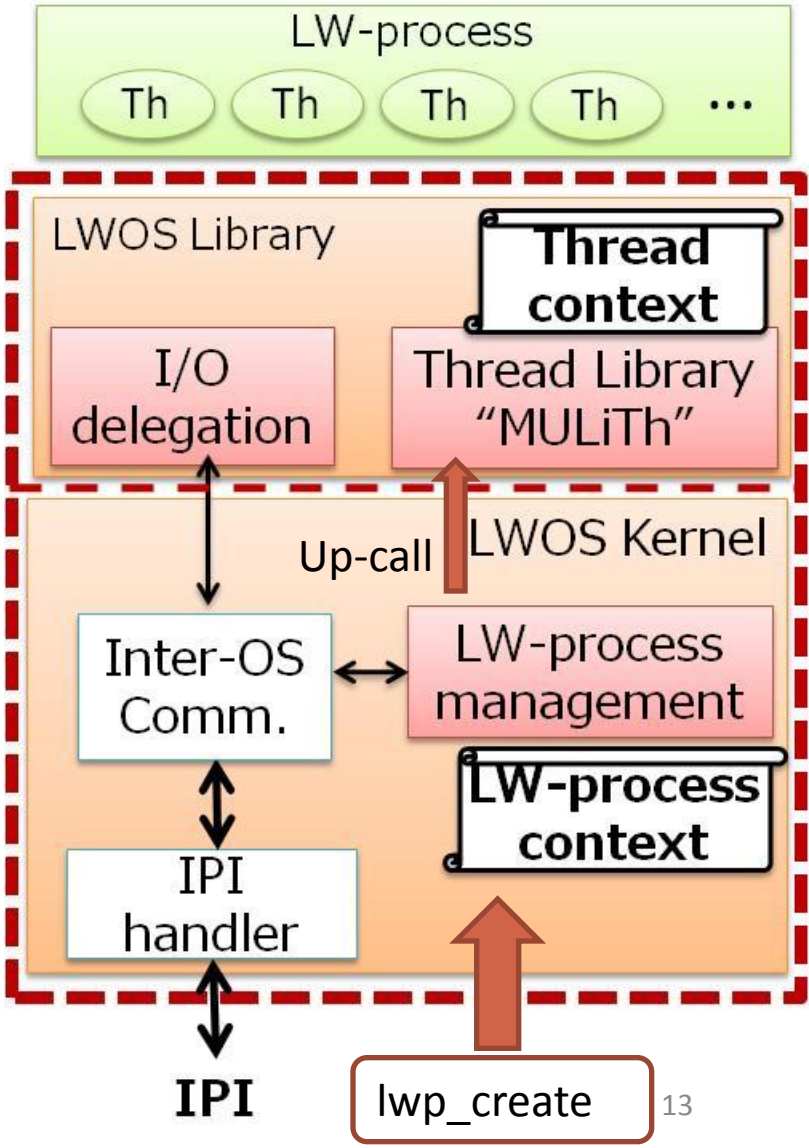


LWOS structure (Kernel Level)

- LW-process Management
  - ➢ Receive LW-process control requests from Host OS

    (Create, Destroy, Suspend, Resume)
    - start the LW-process execution
    - Up–call to destroy/suspend/resume
  - ➢ Keep the LW-process context
    - The number of physical core and IDs
    - the start address of the page table

    (set it to the page directory base register)
    - the Inter-OS comm. buffer address
    - the entry point address of the program



Inter-OS communication

- Set the command items and arguments to the inter-OS communication buffer at User-level
  - I/O access datas are directly written to LW-process address space
- Send “IPI” interruption at Kernel-level

→ reducing the buffer copy overhead

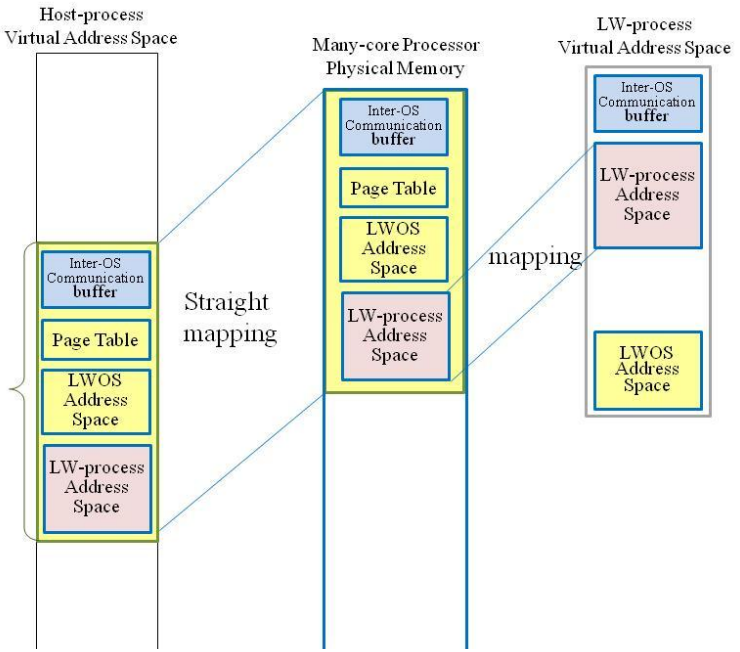


Table 1: LW-process management notification items.

Items	Send to	Type
LW-process Create	LWOS	Synchronous
LW-process Suspend	LWOS	Synchronous
LW-process Resume	LWOS	Synchronous
LW-process Stop	LWOS	Synchronous
LW-process State Notify	Host OS	Unidirectional

Table 2: Memory management norification items.

Items	Send to	Type
Page Fault Notice	Host OS	Synchronous
Page assign Notice	Host OS	Asynchronous

Table 3: I/O access notification items.

Items	Send to	Type
Device Open	Host OS	Asynchronous
Device Close	Host OS	Asynchronous
Device Read	Host OS	Asynchronous
Device Write	Host OS	Asynchronous

LW-process management API

- Host OS APIs
  - Host process uses these APIs for a *LW-process control*

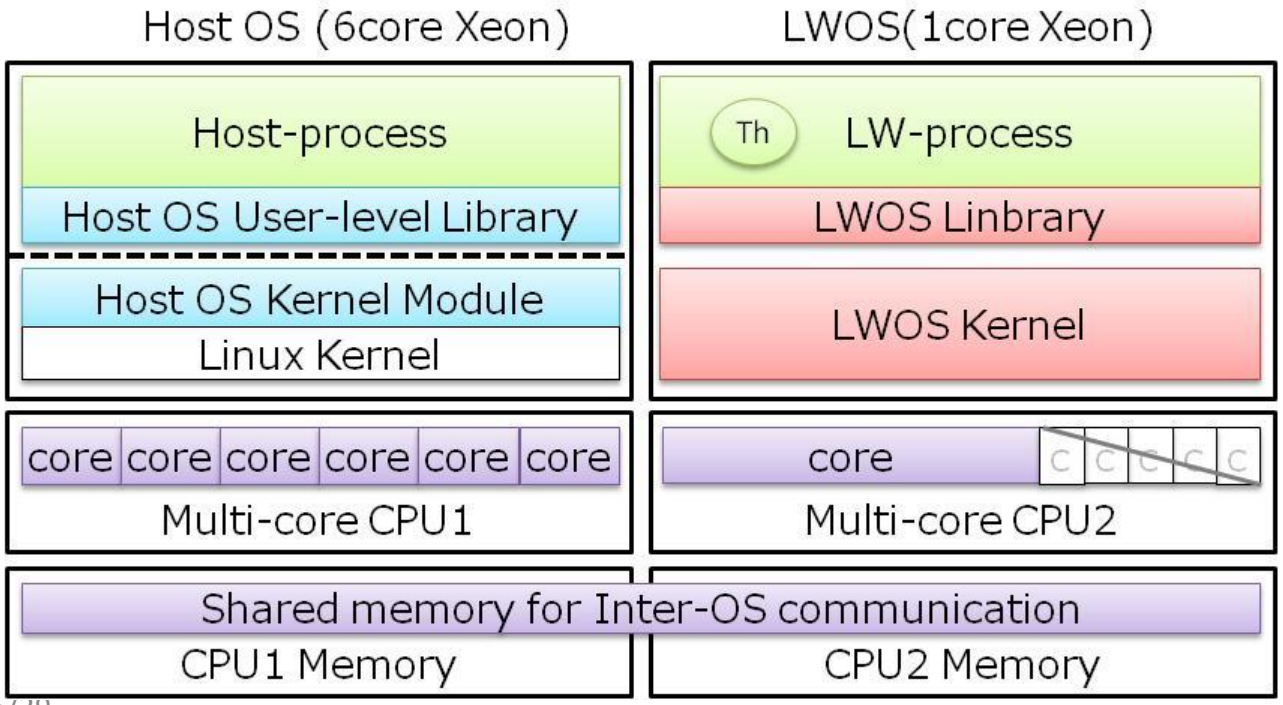
Function name	Specification
lwp_create, lwp_suspend, lwp_resume, lwp_destroy	LW-process control (to LWOS)
lwp_wait	Wait the LW-process exit (from LWOS)

- LWOS APIs
  - *Memory and thread management* API are performed in LWOS without notifying to HOST OS.

Function name	Specification
exit	LW-process exit (to Host)
brk, sbrk	Management of heap memory
open, close, read, write, ioctl	Linux File I/O (to Host)
pthread_*	POSIX thread I/F

Prototype system for evaluation

- Two Xeon CPUs performed Multi-core and Many-core.
  - Confirm processing mechanism of LW-process creation
  - LWOS for Many-core CPU is under development
    - LWOS is executed on one core.
    - LW-process and LWOS library are implemented to kernel level.



- LinuxKernel 2.6.18
- Intel Xeon X5690 **3.46G**Hz **6**core **x2**
- DDR3-1333 **12G**B **x2**

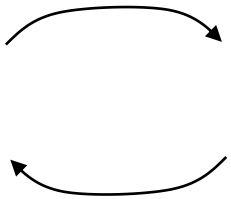
Evaluation of LW-process creation

- measure the execution time of a program that creates the LW-process and then immediately terminates it.
  - compared with Linux fork-exec overhead (only advisory)

environment	time	Ratio
Proposed hybrid OS	**110us**	**147%**
Linux	75us	100%

Host-process Code

```
void  main() {
    lwpid = lwp_create( )
    lwp_wait( lwpid );
}
```



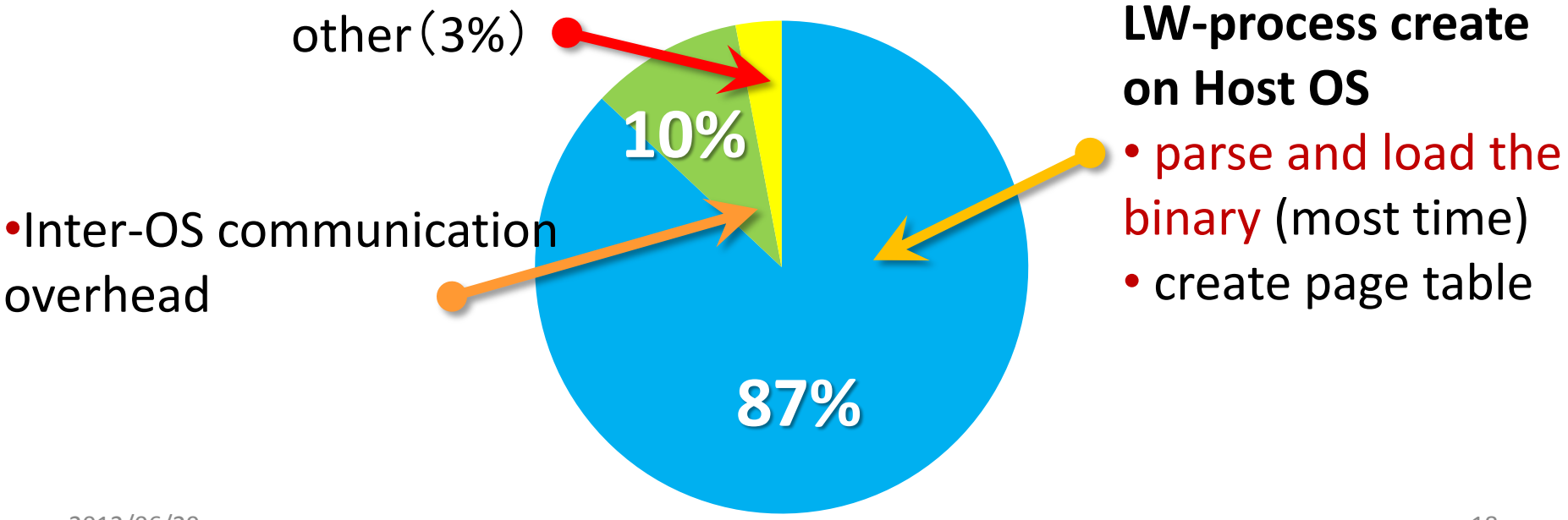
LW-process Code

```
void main() {
  exit();
}
```

bynary size = 1KB

Analysis of the processing time

- The parse and load of the binary file use most time.
- The rate of Inter-OS communication overhead was low.
  - ➢ we used internal bus on Xeon system in this evaluation.
    - communication speed is faster than real MIC environment.
  - ➢ we will evaluate the processing rate on MIC environment.



Conclusion

- We have proposed a system in which the functions for managing the resources of many-core processors are delegated to the Host OS running on multi-core processor in a parallel computing system that uses both multi-core processors and many-core processors.
- That approach allows the LWOS that runs on a many-core processor to be dedicated to the execution of parallel computation programs.
- We have described the structures of the Host OS and LWOS and explained about inter-OS communication.

Future works

- LWOS implementation on MIC processor
  - Thread management on many-core processor
  - Evaluate LW-process management overhead and inter-OS communication overhead
- We will investigate many-core assignment algorithms on Host OS to improve the parallel computing performance of many-core processors.

Thank you

LW-process control APIs

lwp_create

- Binary file path, the number of core, Stack size, heap size
- return : process ID

lwp_suspend / lwp_resume

- process ID

lwp_destroy

- process ID

lwp_wait

- process ID

```
void api_sample(void)
{
    /*
     * 4core, 1MB stack, 100MB heap
     */
    lwpid = lwp_create (
            "~/calc_program.bin",
            4,
            0x100000, 0x6400000 );

    /* suspend and redume*/
    lwp_suspend ( lwpid ) ;
    lwp_resume ( lwpid );

    /* wait for exit*/
    lwp_wait ( lwpid );
}
```