An Efficient Inter-node Communication System with Lightweight-thread Scheduling

HPCC'19 (August 10-12, 2019)

Takuya Fukuoka, Wataru Endo, Kenjiro Taura

The University of Tokyo

Abstracts

- With many cores in one chip, there are increasing needs to exploit inter/intra-node parallelism efficiently
- We propose a new communication system MPI+myth, which enables efficient overlapping of inter-node communication and intra-node computation with little burden on programmers
- MPI+myth is implemented using MPI and a user-level thread library, MassiveThreads

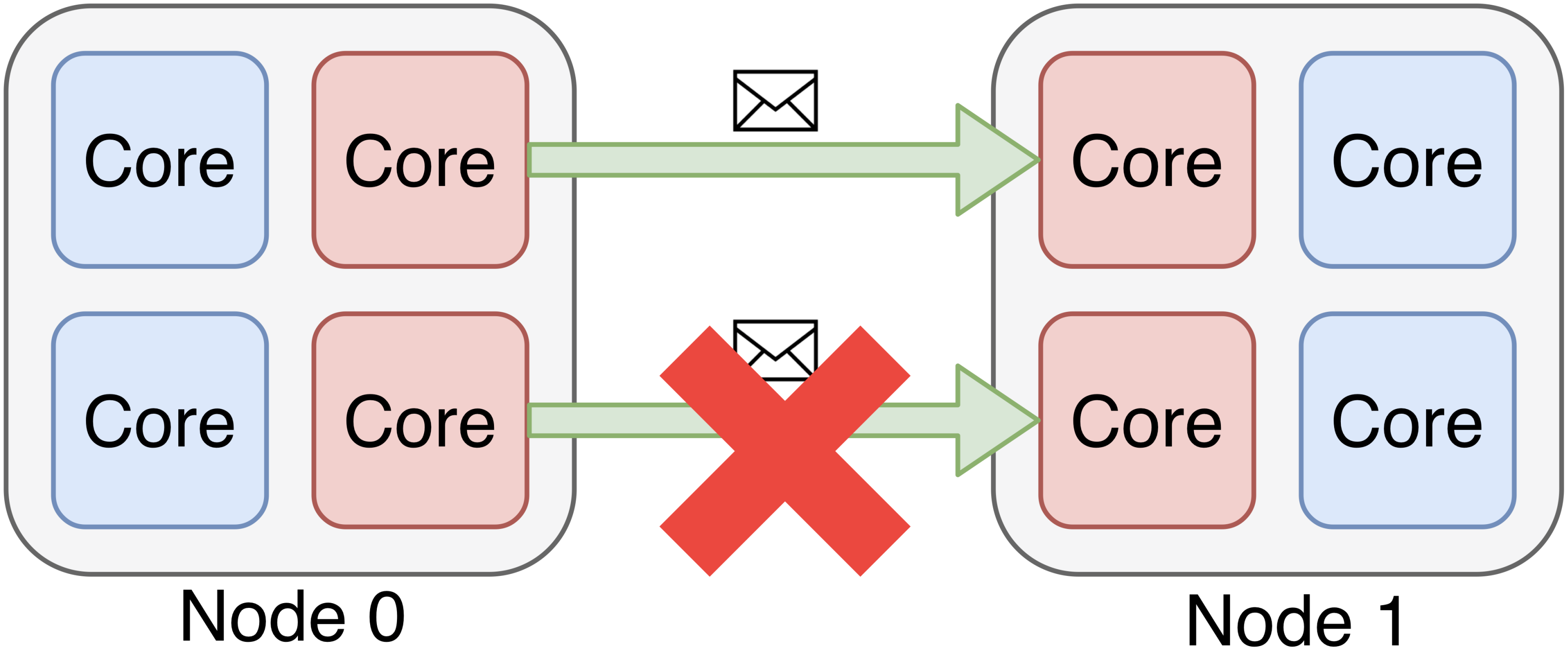
Background

MPI (Message Passing Interface)

- An interface for communication between nodes in a cluster
- Two kinds of APIs: blocking APIs and non-blocking APIs
- It is normal to use blocking APIs because of its simplicity

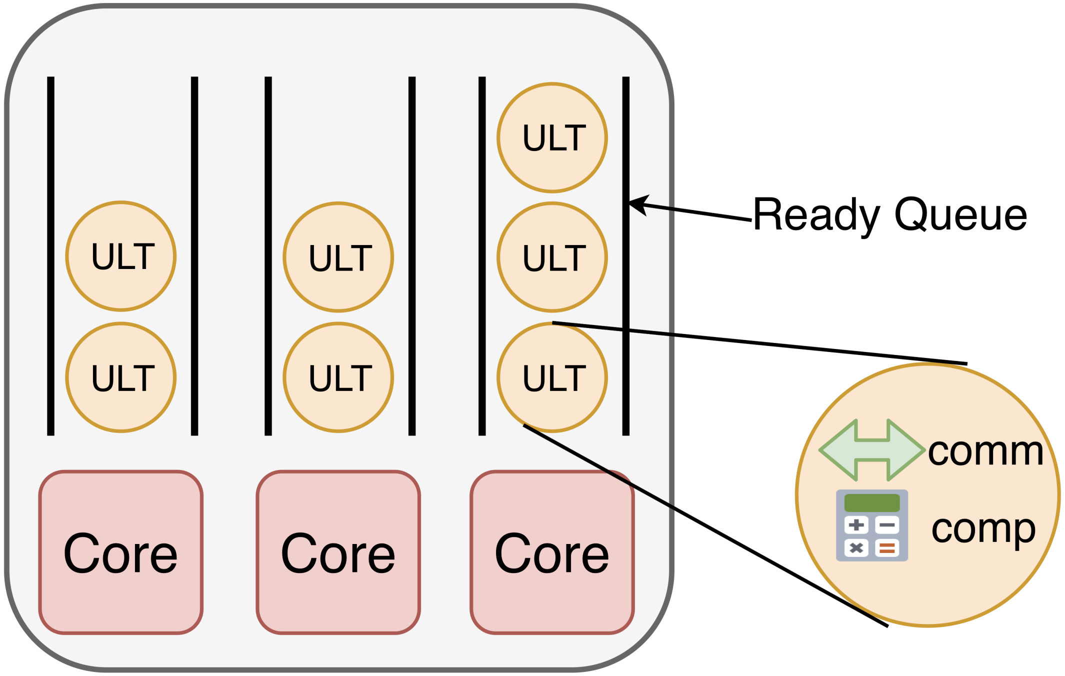
```
// Example usege of non-blocking APIs
MPI_Request req;
int flag;
MPI_Isend(..., &req);
computation();
do{
    MPI_Test(req, &flag, ...)
}while(!flag)
```

The Limitation of multi-threaded invocation of MPI



- In a normal mode, you cannot invoke MPI from multiple threads
- In order to invocate MPI from multiple threads, you have to use a special mode (MPI_THREAD_MULTIPLE)
  - It is known to perform poorly because of its complex and heavy mutual exclusion.

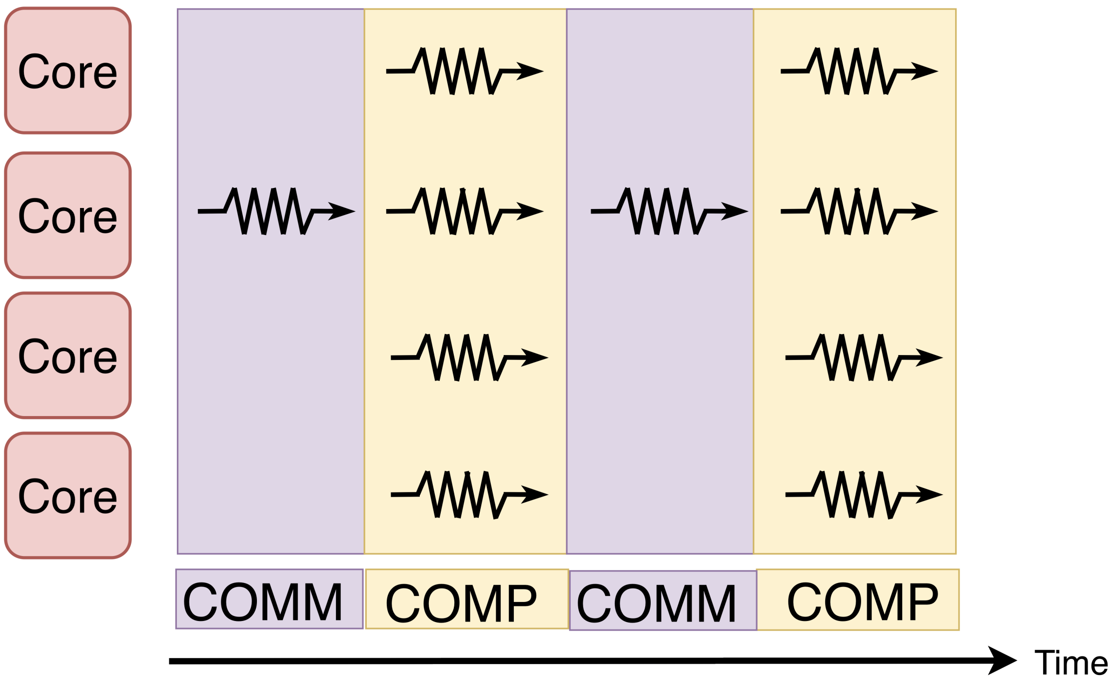
User-level Thread (ULT)



- A ULT is a thread implemented in user space and multiple ULTs can be mapped into one Kernel-level thread (KLT)
- Compared with KLTs, ULTs can be lightweight and thread creation and context-switching can be done at lower cost.
- ULTs are scheduled through ready queues in which executable ULTs can be enqueued and dequeued

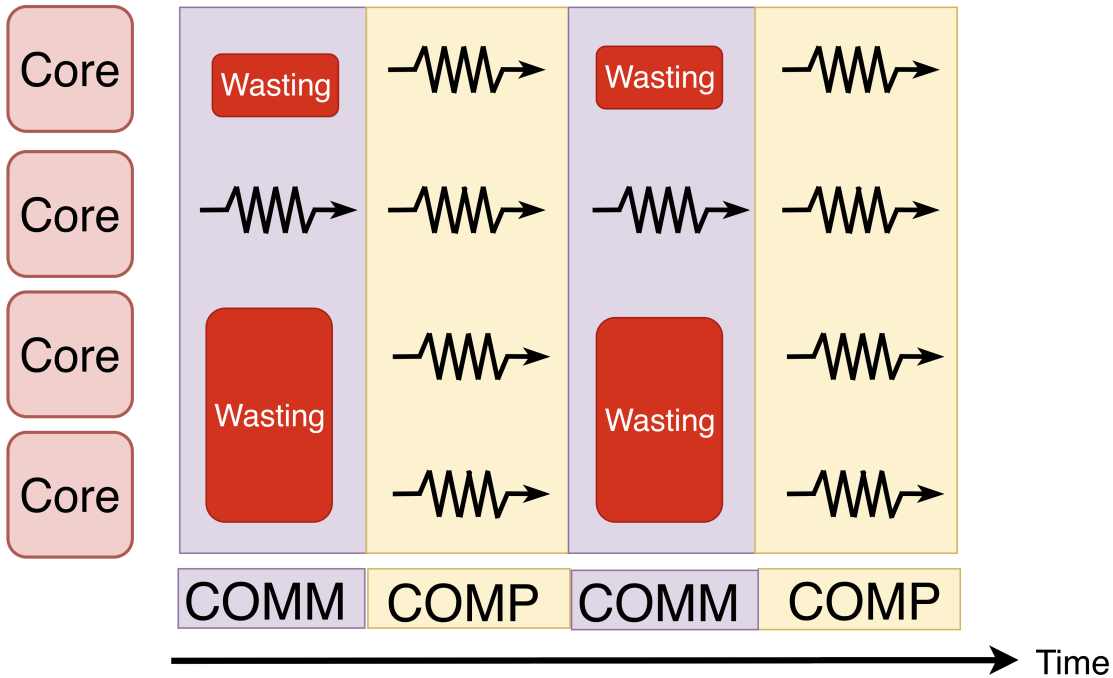
Introduction

Conventional Method for Parallelization



- Separating phases: communication phases and computation phases
- In communication phases, only a master thread issues MPI functions
- In computation phases, each threads executes computation through shared memory

A Problem in the Conventional Method



- Waste of resources of cores which are not in charge of the master thread in communication phases
- In order to use these cores efficiently, programmers have to describe overlapping of communication and computation
  - It requires considering the dependencies
  - It can be heavy burdens on programmers

Ideal Description of Parallelization

- Programmers create many ULTs (tasks) in which communication and computation is issued freely
    - No needs to separate communication and computation phases
- Programmers do not have to use non-blocking APIs of MPI to describe overlapping
    - Little burden on programmers

Obstacles for the Ideal Description

- Multi-threaded MPI invocations are necessary
  - Software Offloading [1] is already proposed to avoid the overhead of MPI_THREAD_MULTIPLE
  - Delegate all communication to one thread
- Just combining SoftwareOffloading and ULT libraries is not enough
  - Other than this, the runtime has to be equipped with a mechanism to overlap communication and computation efficiently

1. Vaidyanathan, K. et al. (2015). Improving Concurrency and Asynchrony in Multithreaded MPI Applications using Software Offloading - SC'15.

Proposed System: MPI+myth

Proposed System: MPI+myth

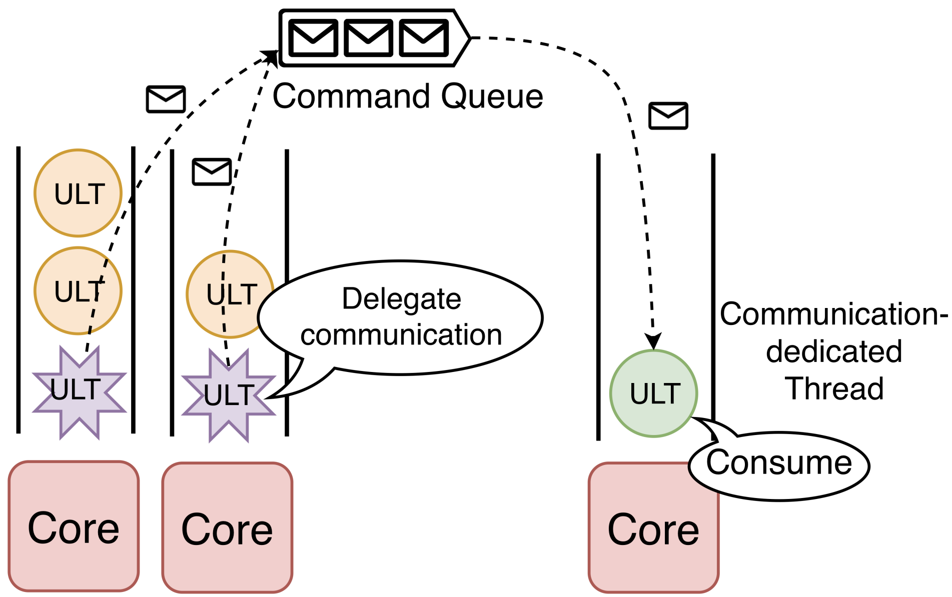
```
func(){
    MPI_Send();
    computation();
}

myth_creare(*func, args, ...)
```

- MPI+myth combines MPI for inter-node communication and ULTs (user-level threads) for intra-node parallelism
- Programmers describe applications using APIs of ULT libraries, MassiveThreads and blocking APIs of MPI
- The runtime has responsibility in overlapping of communication and computation

Two Characteristics of the Implementation of MPI+myth

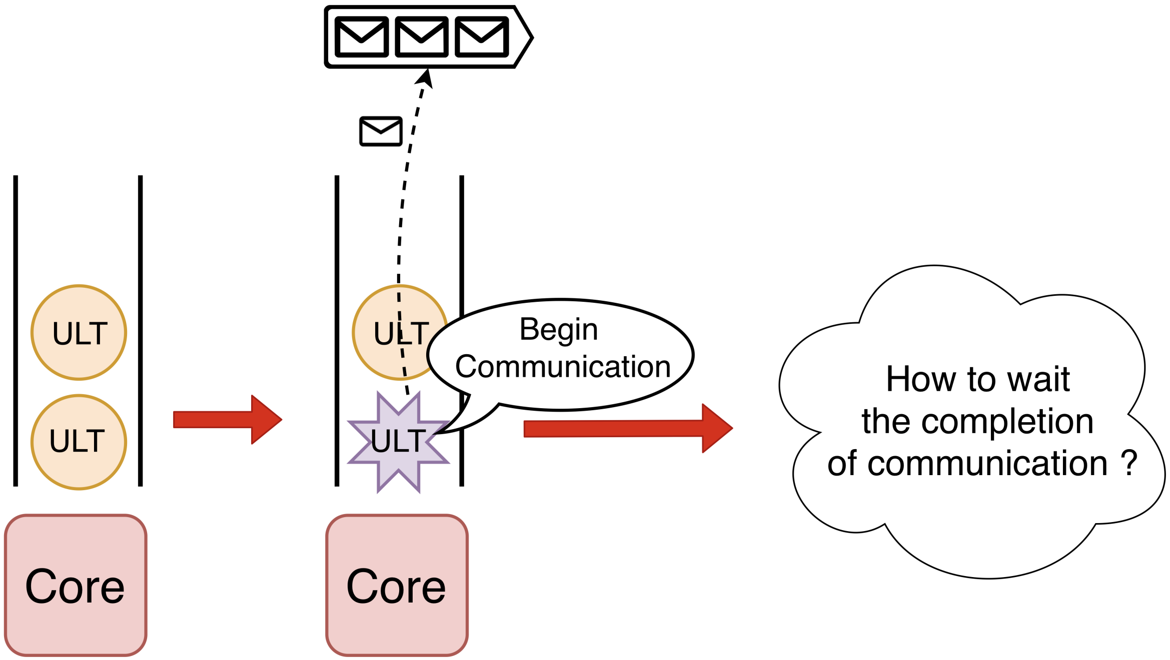
- Improve performance by combining Software Offloading and MassiveThreads and avoiding the use of multi-threaded mode of MPI (MPI_THREAD_MULTIPLE)
- Equipped with a mechanism to overlap communication and computation
    - The runtime detects the communication and communicating ULTs release the core to other ULTs

First Characteristics of MPI+myth: Avoid multi-threaded MPI invocations



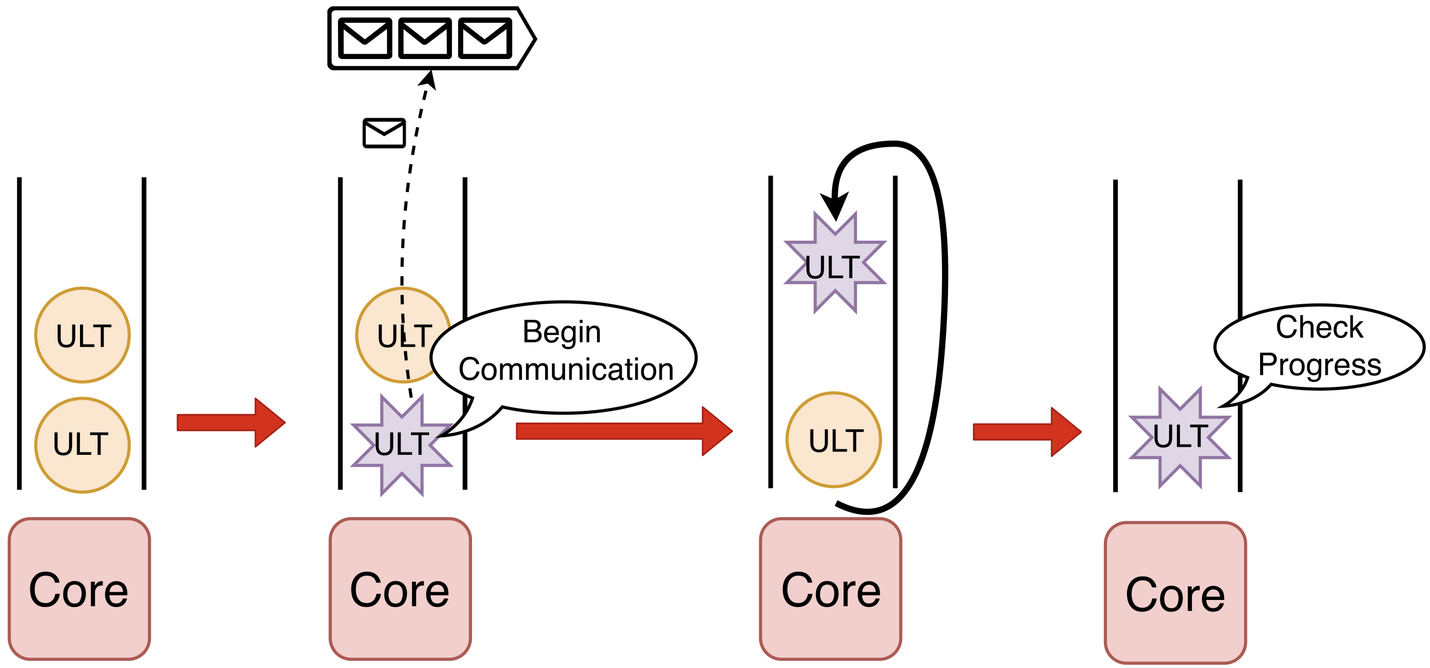
- Introduce a technique called Software Offloading
- Delegate all MPI invocations to a communication-dedicated thread through a command queue [1]
- When a ULT issues communication,its name of a communication function and its arguments are inserted into the command queue

Second Characteristics of MPI+myth: Overlap communication and computation



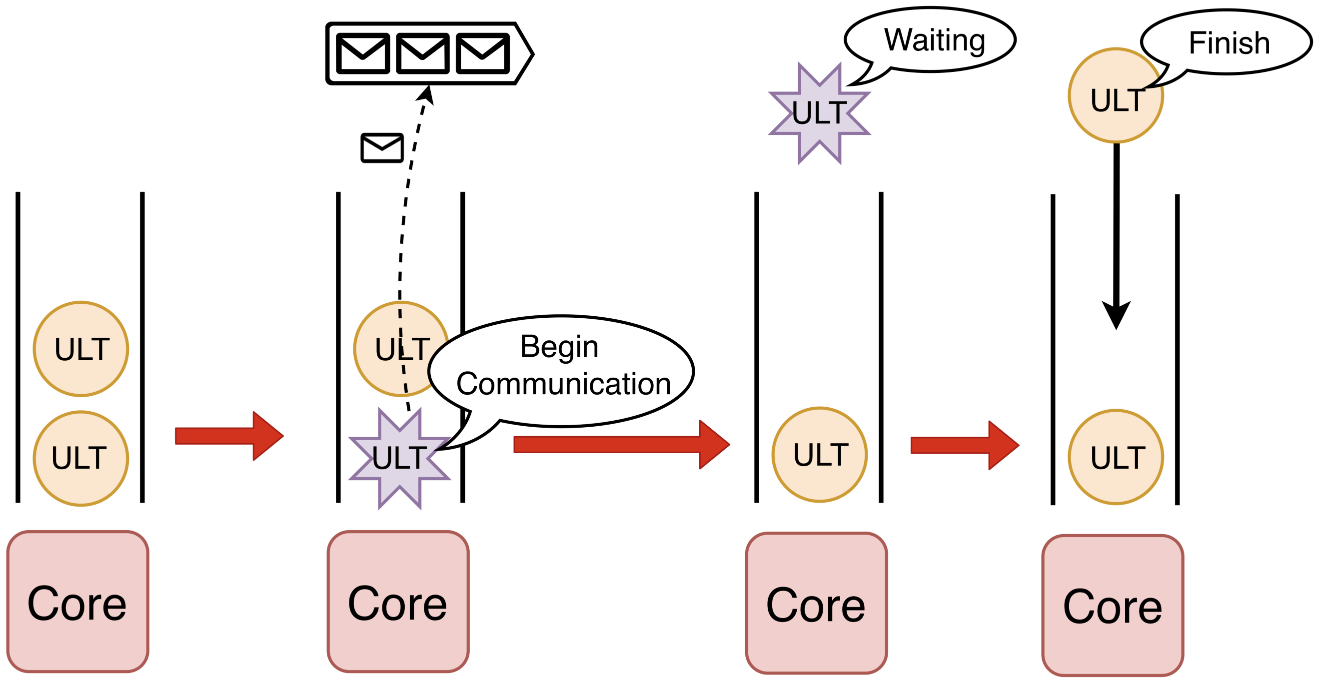
- The runtime is in charge of overlapping communication and computation
- Two kinds of methods for waiting the communication

First Method for Waiting Communication



- First method is to re-insert communicating ULTs to the back of ready queue when the communication continues
  - Communicating ULTs check its progress of communication every time its turn comes
  - Widely used method due to its low implementation cost

Second Method for Waiting Communication



- Second method is to remove communicating ULTs from the ready queue
  - When the communication completes, the ULT is inserted to the back of the ready queue immediately
- Our system adopts the second method because it can avoid a situation in which the ready queue is filled with communicating ULTs

Related Work

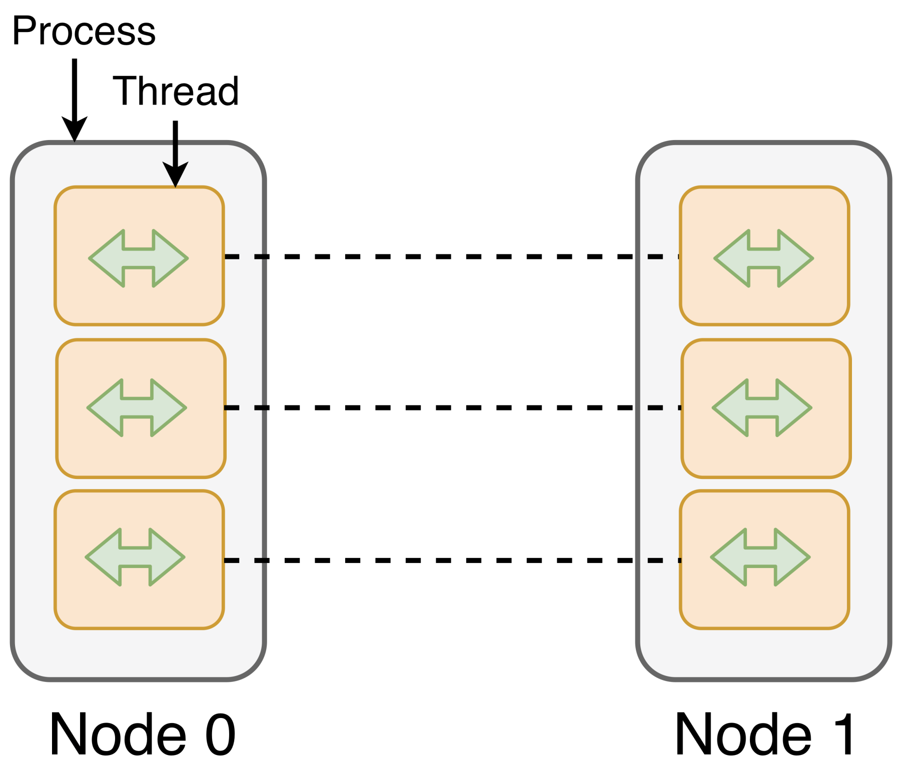
	Avoid Overhead of MPI_THREAD_MULTIPLE	Explicit Description of Overlapping
MPI/SMPSs [2]	unsupported	necessary
HCMPI [3]	yes	necessary
MPIQ [4]	no	unnecessary
MPI+Argobots [5]	no	unnecessary
MPI+myth	yes	unnecessary

2. V.Marjanović, et al. (2010) Overlapping Communication and Computation by Using a Hybrid MPI/SMPSs Approach - ICS
3. S. Chatterjee, et al. (2013) Integrating Asynchronous Task Parallelism with MPI - IPDPS'13.
4. D. Stark, et al. (2014) Early Experiences Co-Scheduling Work and Communication Tasks for Hybrid MPI+X Applications
5. H. Lu, et al. (2015). MPI+ULT: Overlapping communication and computation with user-level threads - HPCC'15

Experimental Environment

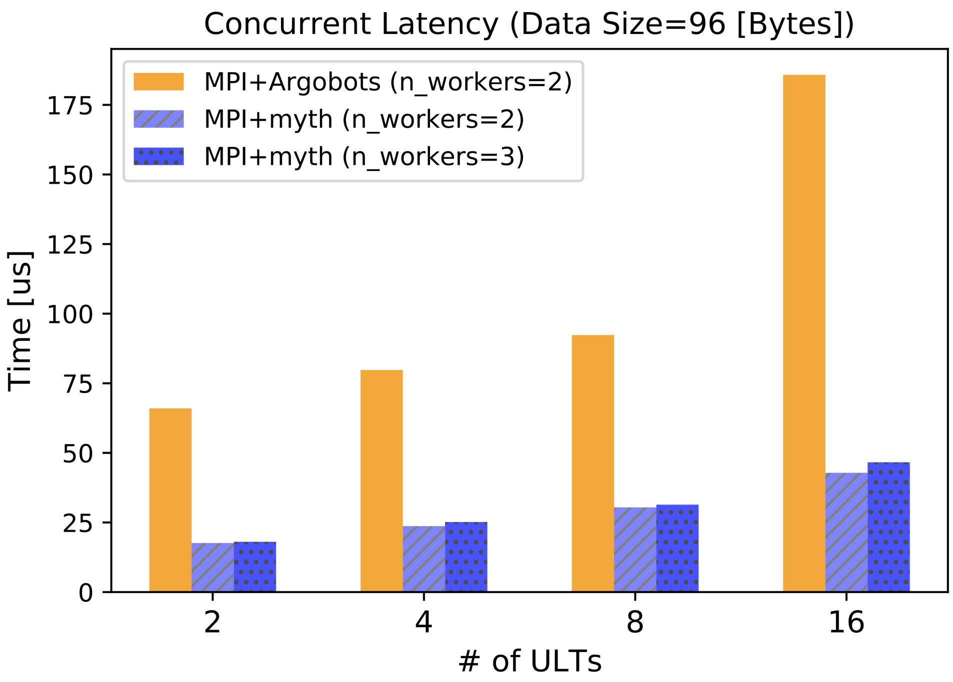
System	Reedbush-U
Interconnect	InfiniBand EDR 4x (100 Gbps)
Processor	Intel Xeon E5-2695v4 (Broadwell-EP)
# of Processors / Node	2
# of cores / Node	36
Memory	256 GB

What is Concurrent Latency ?



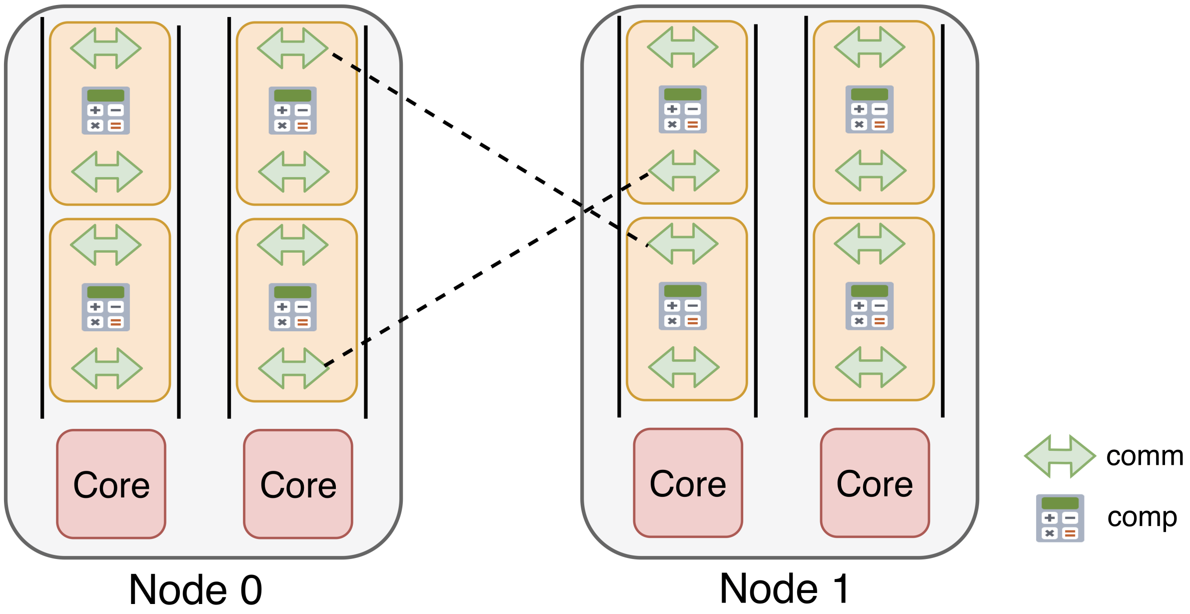
- Two processes are generated and each process generate threads
- Each thread communicate with a thread in another process

Concurrent Latency



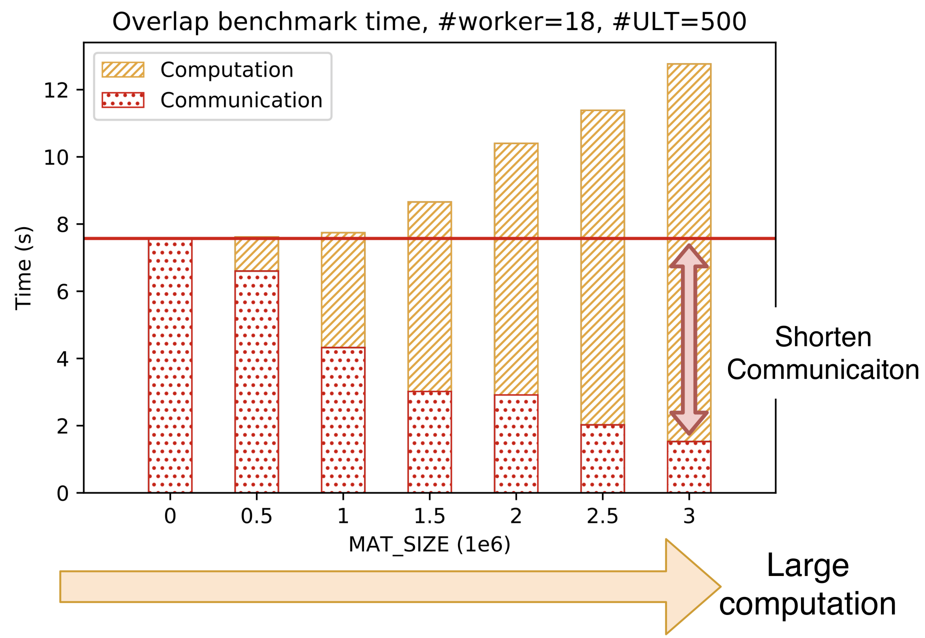
- Compare our system with MPI+Argobots which combines MPI and ULTs
- Our system occupies one core for a communication-dedicated thread
- Our system performs better than MPI+Argobots because our system can avoid the overhead of MPI_THREAD_MULTIPLE

Overlap Benchmark



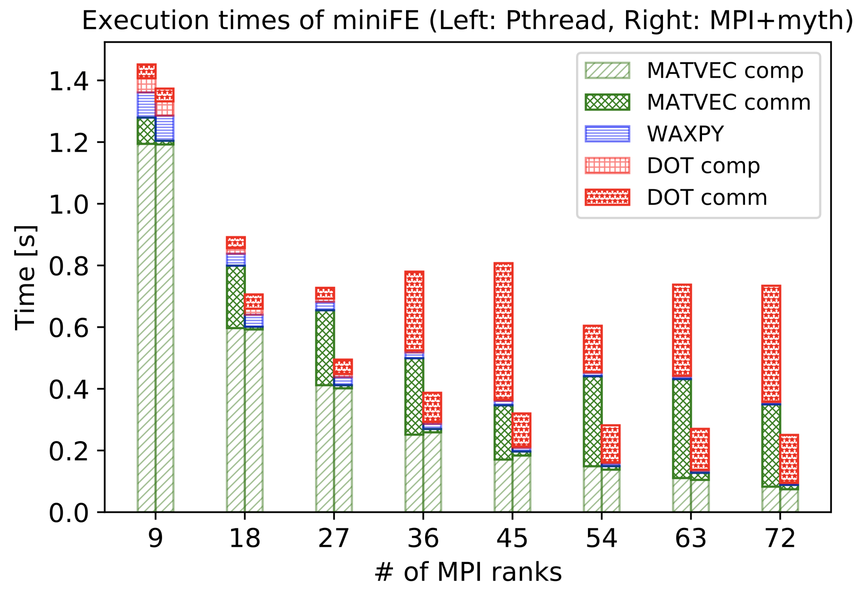
- We create two MPI processes and each process creates multiple ULTs
- One ULT executes blocking communication of 1MB, computation, and blocking communicaion of 1MB again
- The number of cores is 18 and the number of ULTs is 500

Overlap Benchmark Result



- Fix the amount of communication (1MB) and change the amount of computation (which is in proportion to MAT_SIZE in the figure) and measure the whole time of benchmarks
- The time for computation is measured with no communication settings, and the time for communication is calculated as the difference of the whole time and the computation time.
- The time for communication is shorted, which means overlap is achieved

Application Benchmark



- Compare the time of miniFE [6] which is an mini-app for finite element
- In order to exchange the data between nodes, the same number of threads as the number of neighbor nodes are created
- Our system achieves shortening of communication time by introducing Software Offloading

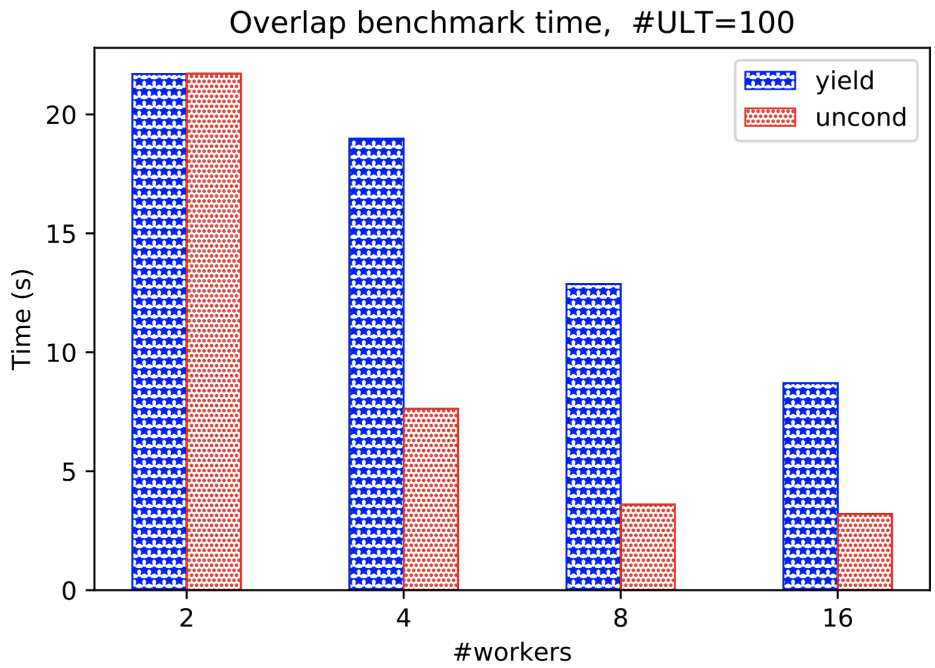
6. https://github.com/Mantevo/miniFE

Conclusions

- MPI+myth combines MPI and ULT and it lays little burdens on programmers by avoiding the use of non-blocking communication
- Two characteristices of the implementation of MPI+myth
  - Improve performance by combining Software Offloading and ULT library
  - Equipped with a mechanism for overlapping blocking communication and computation with efficient waiting method
- MPI+myth was faster than existing parallel systems by between 2.4 to 5.1 times

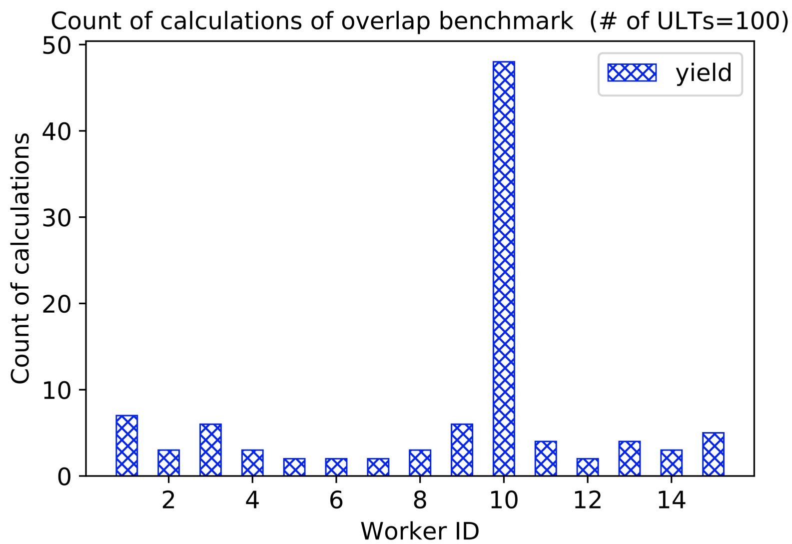
Appendix

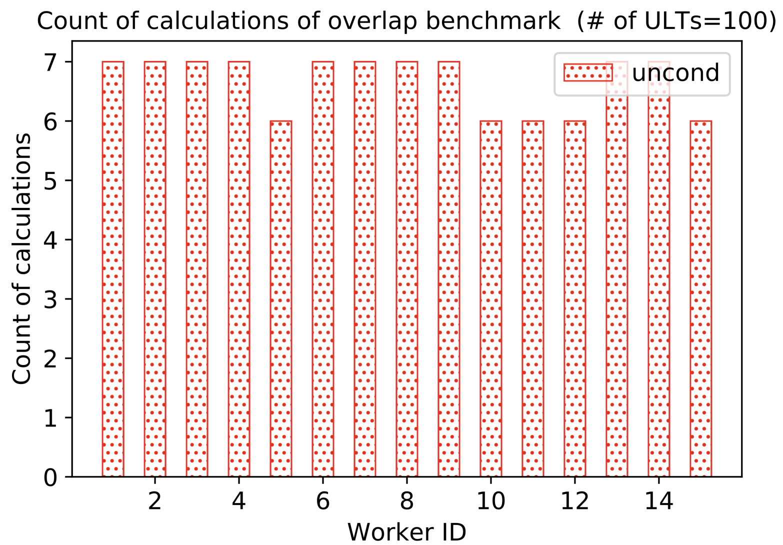
Comparison of Waiting Methods



- The blue bar shows the time with first waiting method and the red bar shows the time with second waiting method, which removes the blocked ULT from a ready queue
- Second waiting method performs better

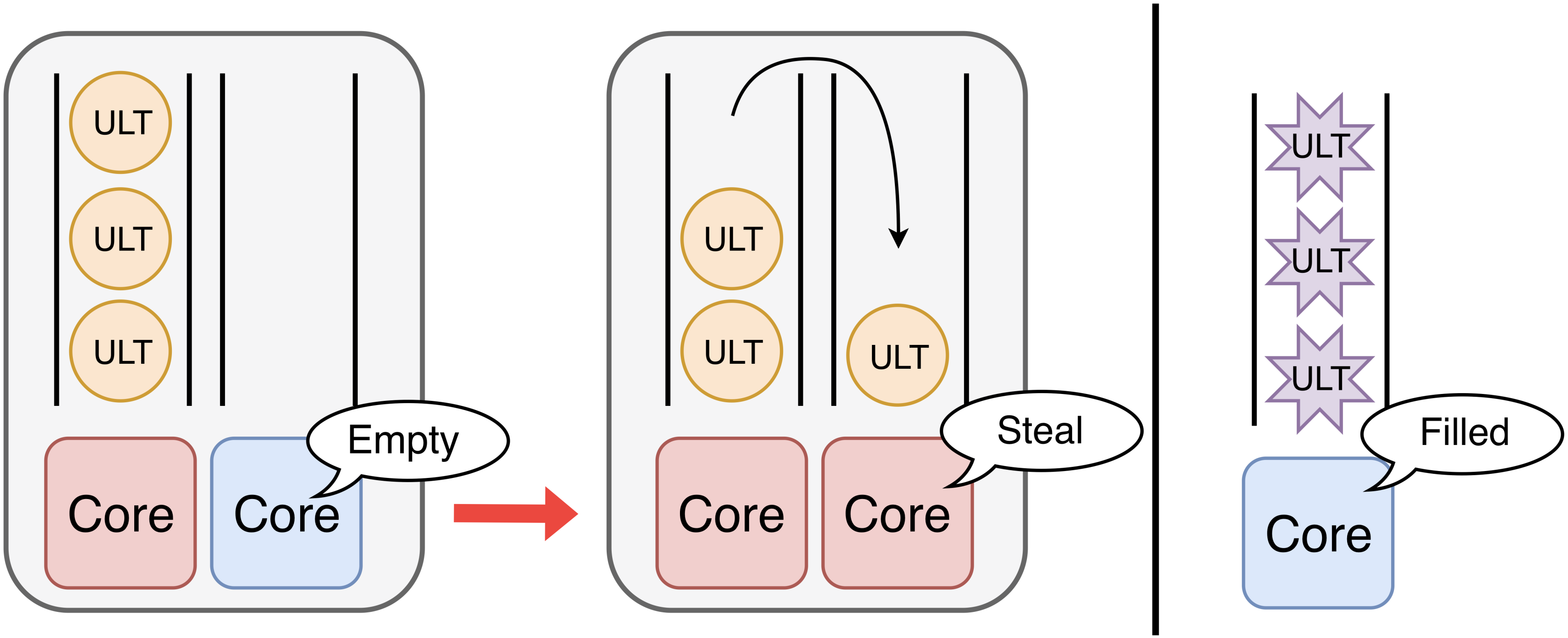
What makes that difference between two scheduling techniques?





- Count the number of computational parts of ULTs each core processed
- With first waiting method, load balancing does not work well

Why first waiting method can disturb efficient load balancing?



- When a core has no ULTs in its ready queue, it can steal a ULT from other cores (left figure)
- When a ready queue is filled with blocked ULTs, stealing does not work (right figure)