# Optimistic Threading Techniques for MPI+ULT


THE UNIVERSITY OF TOKYO  Argonne NATIONAL LABORATORY

Shintaro Iwasaki†1 Abdelhalim Amer‡2 Kenjiro Taura†3 Pavan Balaji‡4

† The University of Tokyo, ‡ Argonne National Laboratory

Contact
1. iwasaki@eidos.ic.i.u-tokyo.ac.jp
2. aamer@anl.gov
3. tau@eidos.ic.i.u-tokyo.ac.jp
4. balaji@anl.gov

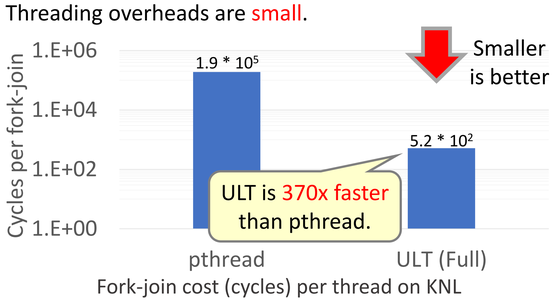

## Introduction

### MPI+Thread

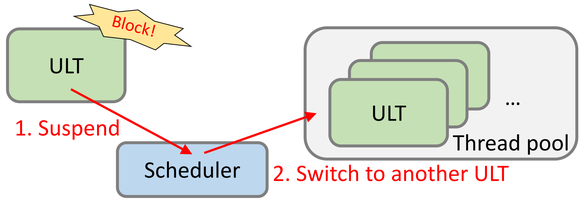
- Hybrid parallelism is common.
  - e.g., MPI+OpenMP.
- OS-level threads (e.g., Pthread) are heavy to exploit fine-grained parallelism..

### MPI+ULT (User-level thread) [1]

- Threading overheads are small.

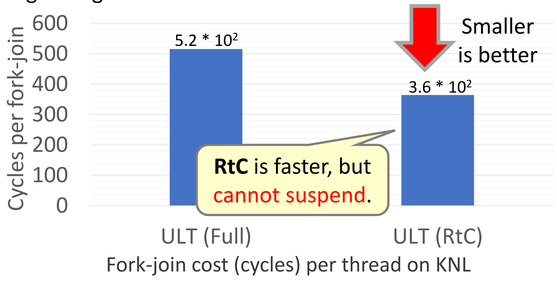



- Efficiently overlap communications and computations.
  - Avoid internal locks in MPI runtime systems by lightweight context switches.




### Problem : Cost of "Suspendability"

- Threads must be created as "suspendable" (**Full**).
- Unsuspendable threads (**RtC**) are known to be more lightweight.




- Few threads suspend in many cases.
  - Not all threads call MPI functions.
  - Suspension is unnecessary when locks are successfully acquired.

We need to create **Full** ULTs if it might call MPI functions, imposing overheads.

→ Can we reduce overheads when we know most threads never suspend?

Full = LCtx, Repeat creating 256 threads & joining all of them; no suspension. / pthread: 500 iterations & 10 warm-ups / Full, RtC: 5000 iterations & 100 warm-ups Average of 10 times execution / Intel Xeon Phi 7210 (KNL) 1 core / 1.3GHz (turbo-boost is off) / Compiled by icc 17.2.174 / Stack size : 16KB / Error < 5% / Red Hat 4.8.5-16 / Huge page is enabled.

Goal:
As fast as **RtC** when threads do not suspend, but able to suspend as well as **Full**.

Basic idea : Dynamic promotion from **RtC**.

## Threading Techniques

- Essence of ULT
  save and restore the context of computation.
- The context is represented as
  1. register values and 2. function stack.
  - Register values are saved by "user-level context switch."

Summary of threading techniques

| | Not suspend | | | Suspend | | |
|---|---|---|---|---|---|---|
| | Change Stack? | Context switches? | Overhead | Rerun sched.? | Overhead | Constraints |
| Full | Yes | Yes | High | No | Low | No |
| SA | No | No | ⋁ | Yes | High | * |
| RtC | No | No | Low | - | - | ** |

* Schedulers must be reentrant. Stack size of schedulers and ULTs is shared.
** Threads are unable to yield.

### Fully-Fledged Thread (**Full**)

- Overheads (no suspension)
  = 2 user-level context switches
  + stack management + scheduling
- Many existing threading libraries adopt this method.
- The overheads is large no matter whether a thread suspends or not.

### Run-to-Completion Thread (**RtC**)

- Overheads (no suspension)
  = scheduling
- Cannot suspend.

Optimistic threading technique

### Scheduler Activation (**SA**)

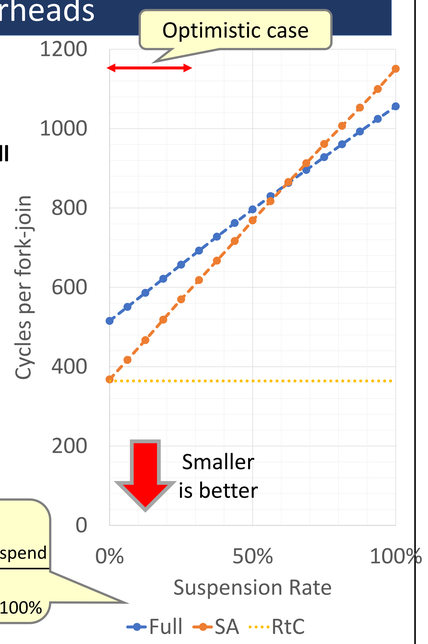
- Overheads (no suspension)
  = scheduling (+ flag management)
- If suspension is needed, it reruns a scheduler from scratch on a new thread.

Constraints
- A scheduler must be reentrant.
- Stack size of schedulers and ULTs must be the same.

- Few runtime systems (Chores[3]) adopted this in the past.

## Fork-Join Overheads

- Implement them in Argobots [2]
- **SA** is faster than **Full** when suspension rate < ~60%.
- **RtC** is fastest when suspension rate = 0, while **RtC** cannot suspend.

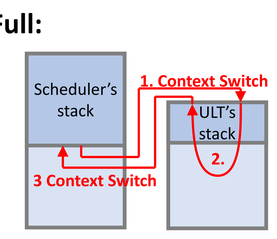



Suspension Rate = (# of ULTs which ever suspend / # of All ULTs) × 100%

Full = LCtx, SA = SA, Repeat creating 256 threads & joining all of them 5000 times (warm-ups: 100); some of them suspend. / Average of 20 times execution / Intel Xeon Phi 7210 (KNL) 1 core / 1.3GHz (turbo-boost is off) / Compiled by icc 17.2.174 / Stack size : 16KB / Error < 5% / Red Hat 4.8.5-16 / Huge page is enabled.

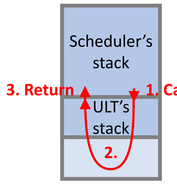
## How do they work? (no-suspension case)

**Full:**



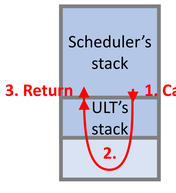

1. A. Save register values of the scheduler.
   B. Save instruction/stack pointers
   C. Invoke a ULT on a newly allocated stack.
2. D. Run a body of a ULT.
   E. Restore register values of the scheduler
3. F. Restore stack pointers
   G. Jump to the instruction pointer of the scheduler.

**RtC:**




1. A. *Call* a ULT
2. B. Run a body of a ULT.
3. C. *Return* to the scheduler.

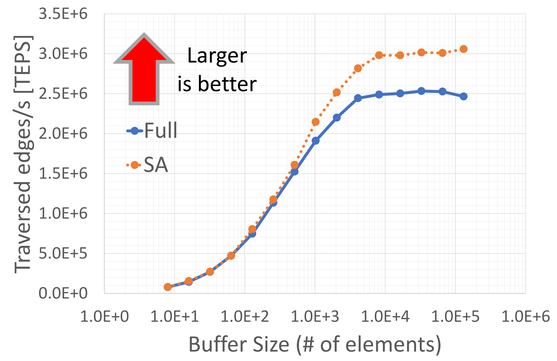
**SA:**

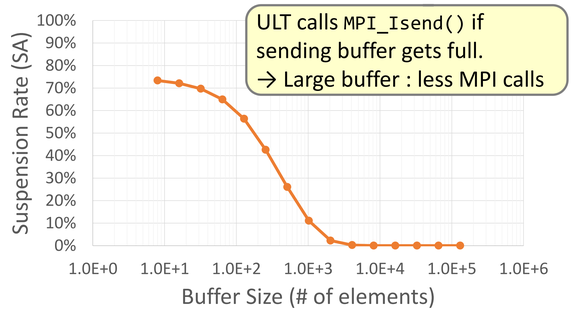



1. A. *Call* a ULT
2. B. Run a body of a ULT.
3. C. *Return* to the scheduler.
   D. Check if a ULT has ever suspended.

## MPI+ULT (Graph500)

- Graph500 parallelized by MPI+ULT.
  - Argobots-aware MPICH 3.2 is used.
- Each ULT is mapped to a single traversal.
- Performance on 4 KNLs (64 workers * 4 MPI processes).




- The optimistic threading technique (**SA**) improved performance.
  - We cannot use **RtC** since ULTs may call MPI functions.

Full = LCtx, SA = SA, Scale factor = 20, edge factor = 16, max # of threads = 4096, finest granularity (traverse per thread), Average of 10 times execution / 4 x Intel Xeon Phi 7210 (KNL) 64 threads / 1.3GHz (turbo-boost is off) / Compiled by icc 17.2.174 / Stack size : 16KB / Error < 5% / Red Hat 4.8.5-16 / Huge page is enabled. / OFI140-psm2

## Future Work

- Alleviate constraints of SA.
- Compare performance with the existing techniques (i.e., OpenMP).


### Acknowledgment

This work is supported by the U.S. Department of Energy, Office of Science, under Contract DE-AC02-06CH11357.


## References


[1] H. Lu, S. Seo, and P. Balaji. MPI+ULT: Overlapping communication and computation with user-level threads. HPCC '15, Aug. 2015.
[2] Argobots: A Lightweight Low-level Threading/Tasking Framework http://www.argobots.org/
[3] D. L. Eager and J. Jahorjan. Chores: Enhanced run-time support for shared-memory parallel computing. TOCS, Feb. 1993.