Casper

An Asynchronous Progress Model for MPI RMA on Many-core Architectures

Min Si[1][2], Antonio J. Peña[1], Jeff Hammond[3], Pavan Balaji[1],
Masamichi Takagi[4], Yutaka Ishikawa[4]

[1] Argonne National Laboratory, USA
{msi, apenya, balaji}@mcs.anl.gov

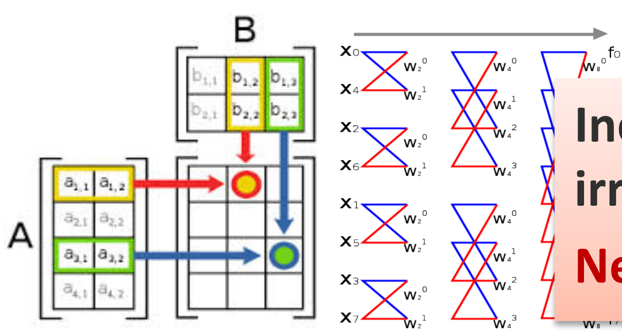
[2] University of Tokyo, Japan
msi@il.is.s.u-tokyo.ac.jp

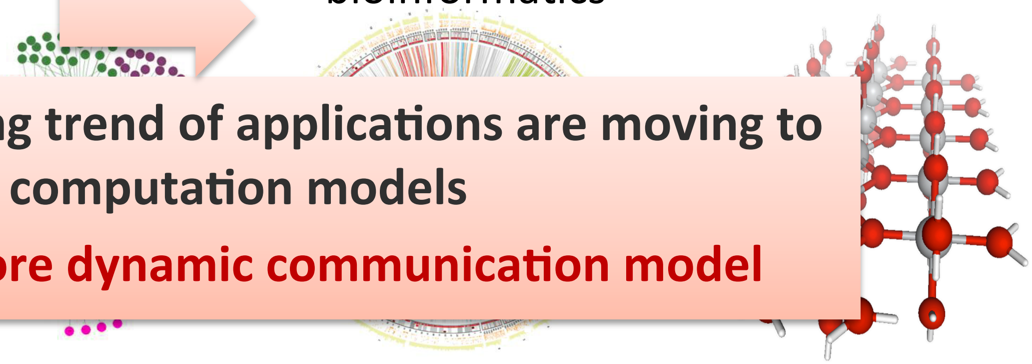
[3] Intel Labs, USA
jeff_hammond@acm.org

[4] RIKEN AICS, Japan
{masamichi.takagi *,yutaka.ishikawa}@riken.jp*

Irregular Computations

- **Regular computations**
  - Organized around dense vectors or matrices
  - **Regular data movement** pattern, use **MPI SEND/RECV or collectives**
  - More local computation, less data movement
  - Example: stencil computation, matrix multiplication, FFT*
- **Irregular computations**
  - Organized around graphs, sparse vectors, more "data driven" in nature
  - Data movement pattern is **irregular and data-dependent**
  - **Growth rate of data movement is much faster than computation**
  - Example: social network analysis, bioinformatics





Increasing trend of applications are moving to irregular computation models

Need more dynamic communication model

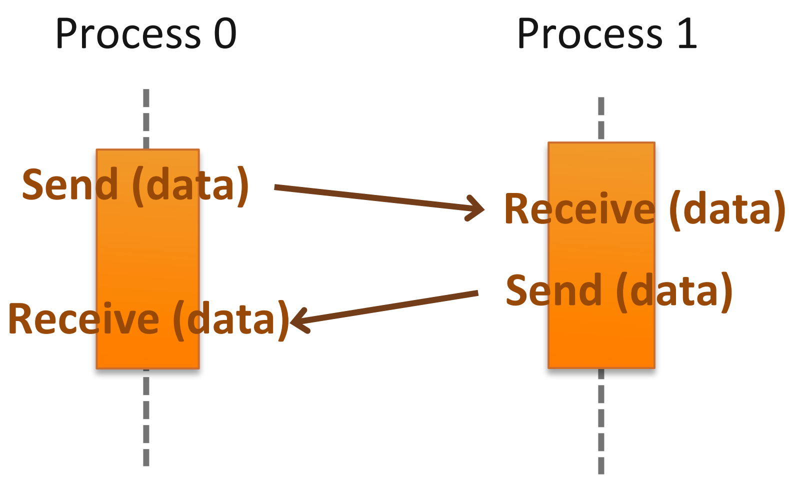
* **FFT** : Fast Fourier Transform

* The primary contents of this slide are contributed by Xin Zhao.

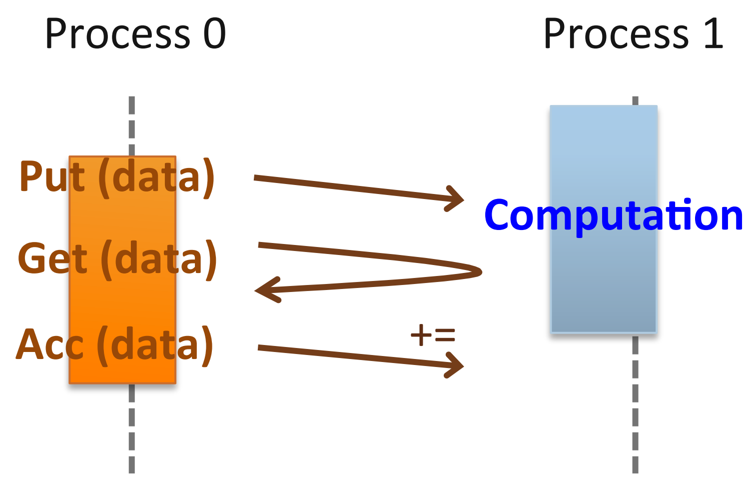
Min Si msi@anl.gov
Argonne National Laboratory, The University of Tokyo

Message Passing Models

- **Two-sided communication**



- **One-sided communication (Remote Memory Access)**



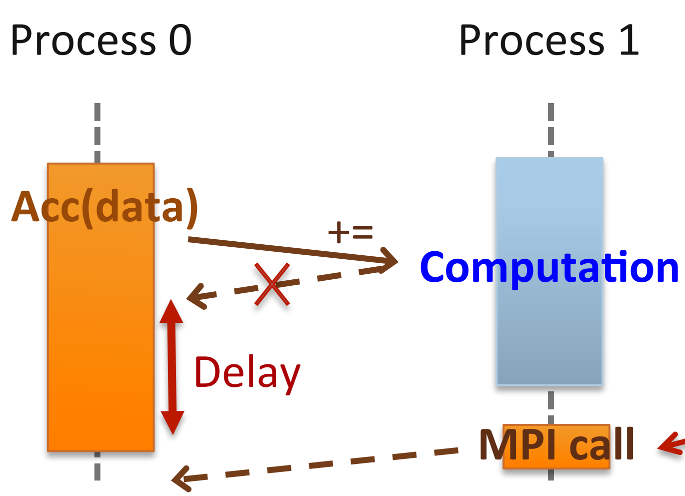
Feature:

- Origin (P0) specifies all communication parameters
- Target (P1) does not explicitly receive or process message

Is communication always asynchronous ?

Problems in Asynchronous Progress

- **One-sided operations are not truly one-sided**
  - In most platforms (e.g., InfiniBand, Cray)
    - Some operations are hardware supported (e.g., contiguous PUT/GET)
    - Other operations **have to be done in software** (e.g., 3D accumulates of double precision data)



Software implementation of one-sided operations means that **the target process has to make an MPI call to make progress**.

Not TRULY asynchronous !

* **RDMA** : Remote Direct Memory Access

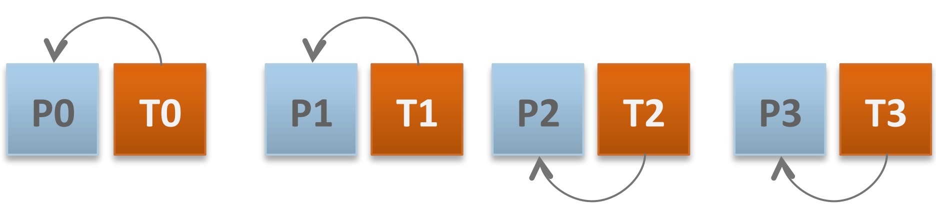
Traditional Approach of ASYNC Progress (1)

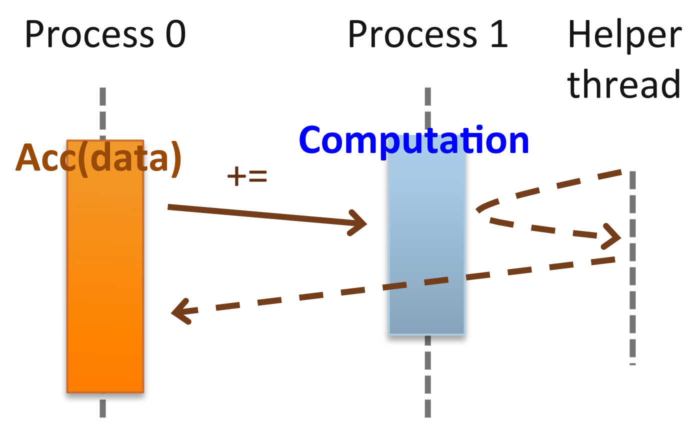
- **Thread-based approach**
  - Every MPI process has a **communication dedicated background thread**
  - Background thread polls MPI progress in order to handle incoming messages for this process
  - Example: MPICH default asynchronous thread, SWAP-bioinformatics

Cons:

× **Waste half of computing cores** or **oversubscribe** cores

× Overhead of **Multithreading safety** of MPI





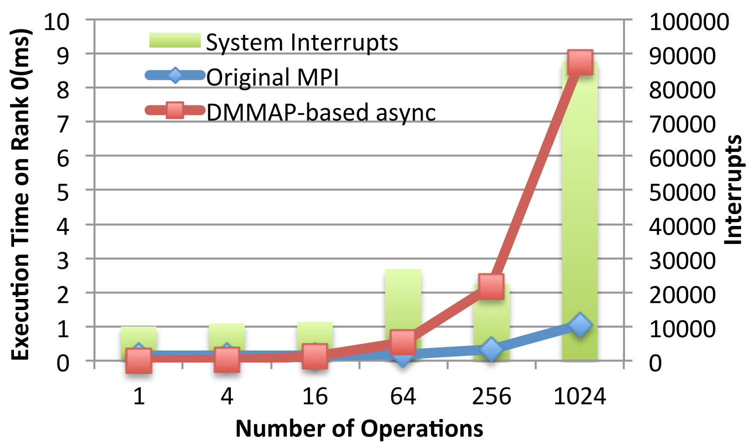
Min Si msi@anl.gov
Argonne National Laboratory, The University of Tokyo

Traditional Approach of ASYNC Progress (2)

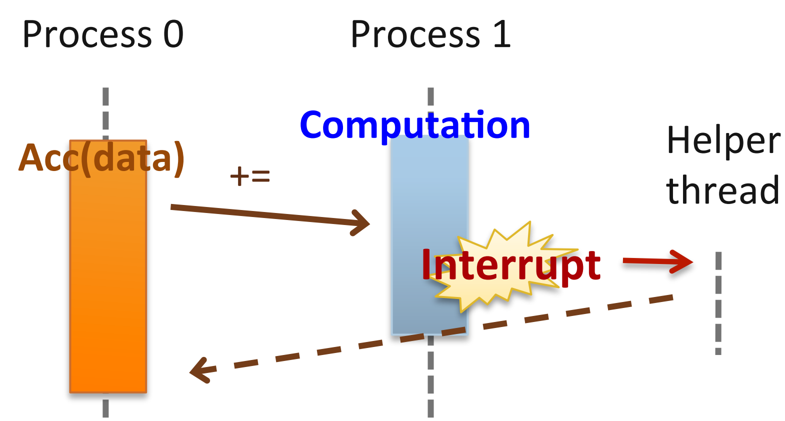
- **Interrupt-based approach**
  - Assume all hardware resources are busy with user computation on target processes
  - Utilize **hardware interrupts** to awaken a kernel thread and process the incoming RMA messages
  - i.e., Cray MPI, IBM MPI on Blue Gene/P

Cons:

× Overhead of **frequent interrupts**



DMMAP-based ASYNC overhead on Cray XC30

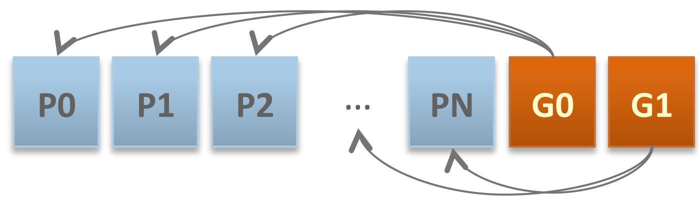


Min Si msi@anl.gov
Argonne National Laboratory, The University of Tokyo

Outline

- Background & Problem statement
- Existing Approaches
- **Our solution : CASPER**
- Ensuring Correctness and Performance
- Evaluation

Casper Process-based ASYNC Progress



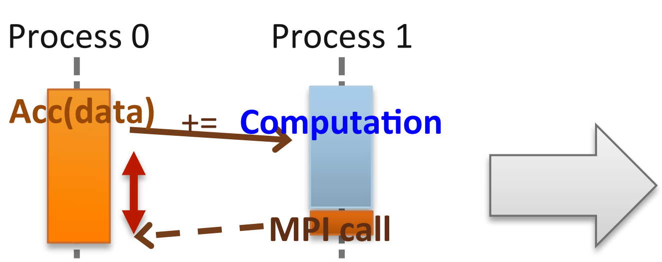
- **Multi- and many-core architectures**
  - Rapidly growing number of cores
  - **Not all of the cores are always keeping busy**
- **Process-based asynchronous progress**
  - Dedicating **arbitrary number of cores to "ghost processes"**
  - **Ghost process intercepts all RMA operations** to the user processes

Pros:

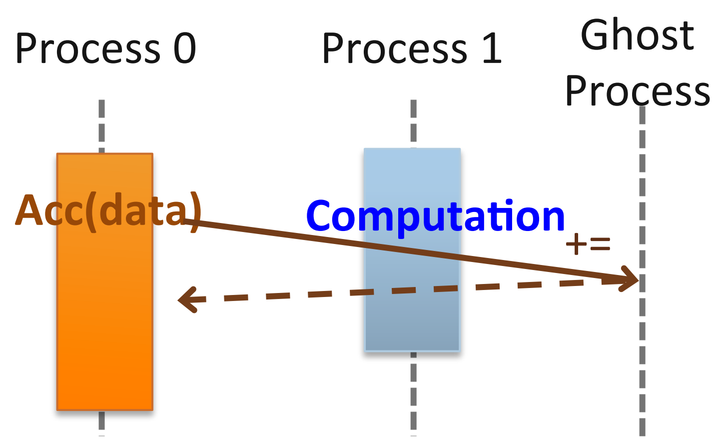
✓ No overhead caused by **multithreading safety** or **frequent interrupts**

✓ **Flexible core deployment**

✓ **Portable PMPI* redirection**



Original communication



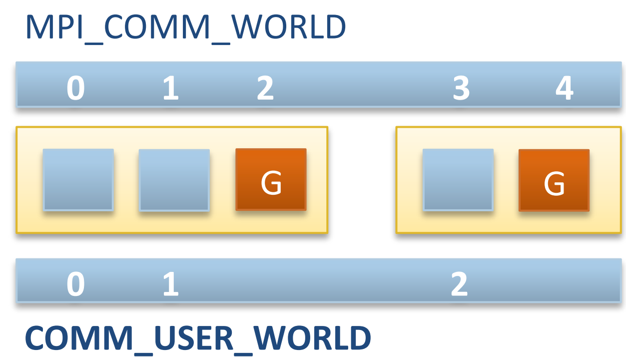
Communication with Casper

* **PMPI** : name-shifted profiling interface of MPI

Min Si msi@anl.gov
Argonne National Laboratory, The University of Tokyo

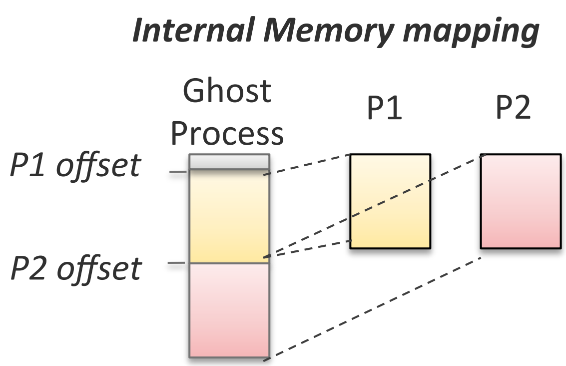
Basic Design of Casper

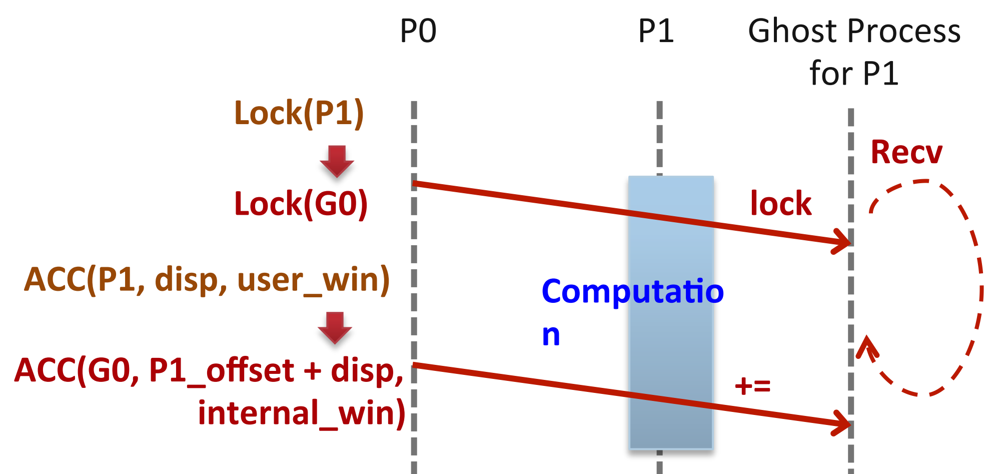
- **Three primary functionalities**
  1. Transparently replace MPI_COMM_WORLD by **COMM_USER_WORLD**



  2. **Shared memory mapping** between local user and ghost processes by using MPI-3 MPI_Win_allocate_shared*

  3. **Redirect RMA operations** to ghost processes



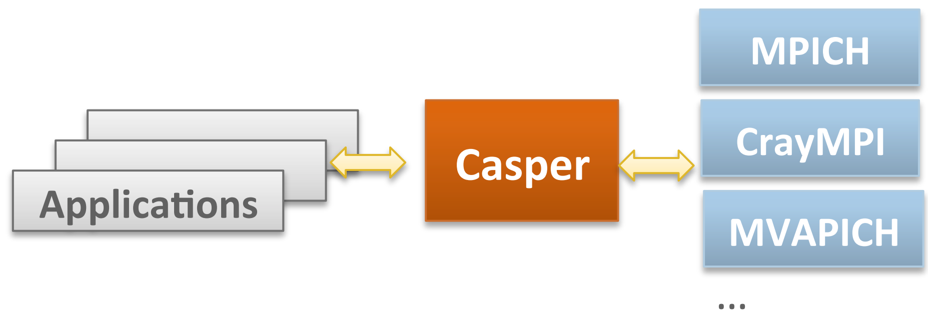


* **MPI_WIN_ALLOCATE_SHARED** : Allocates window that is shared among all processes in the window's group, usually specified with MPI_COMM_TYPE_SHARED communicator.

Ensuring Correctness and Performance

Correctness challenges

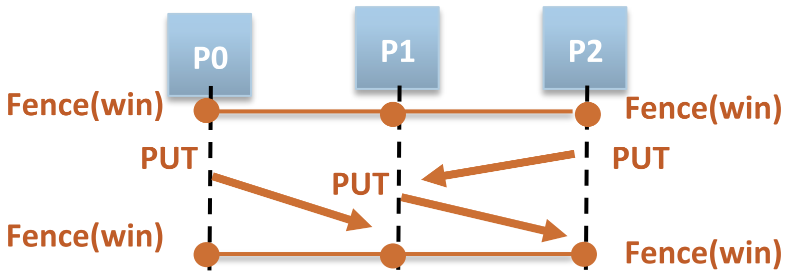
1. Lock Permission Management
2. Self Lock Consistency
3. Managing Multiple Ghost Processes
4. Multiple Simultaneous Epochs



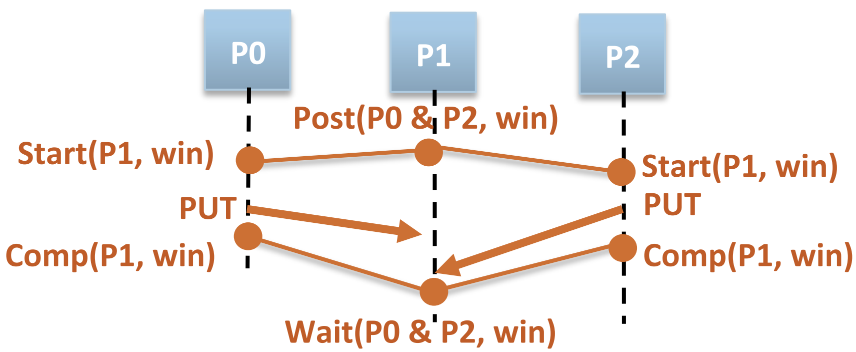
- ✓ Asynchronous progress
- ✓ Correctness
- ✓ Performance

RMA synchronization modes

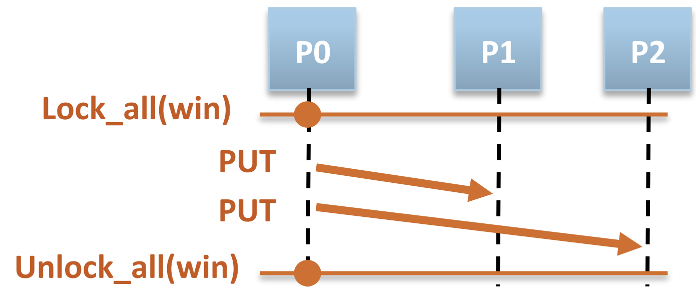
- **Active-target mode**
  - Both origin and target issue synchronization
  - **Fence** (like a global barrier)



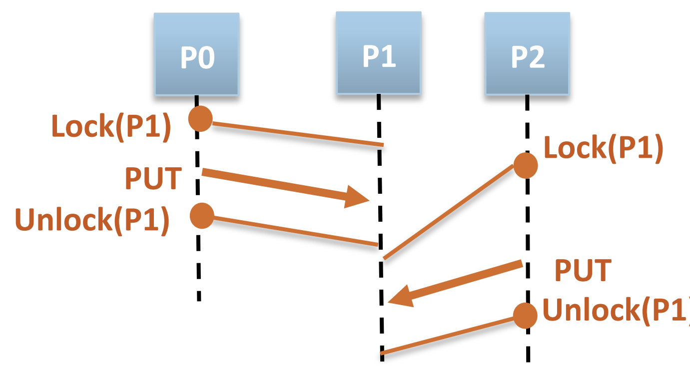
  - **PSCW** (subgroup of Fence)



- **Passive-target mode**
  - Only origin issues synchronization
  - **Lock_all** (shared)



  - **Lock** (shared or exclusive)

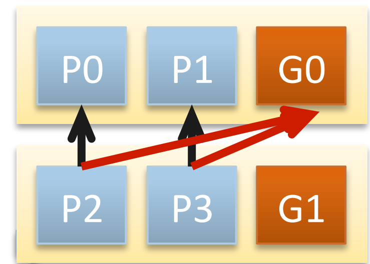


[Correctness Challenge 1]

Lock Permission Management for Shared Ghost Processes (1)

1. Two origins access two targets sharing the same ghost process

[POOR PERF.] Two concurrent lock epochs have to be **serialized**



P2

Lock (P0, win)
Unlock(P0, win)

P3

Lock (P1, win)
Unlock(P1, win)

→

P2

Lock (G0, win)
Unlock(G0, win)

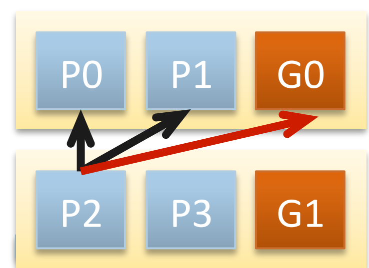
Serialized

P3

Lock (G0, win)
Unlock(G0, win)

2. An origin accesses two targets sharing the same ghost process

[INCORRECT] Nested locks to the same target



Lock (P0, win) Lock (P1, win)
Unlock(P0, win) Unlock(P1, win)

→

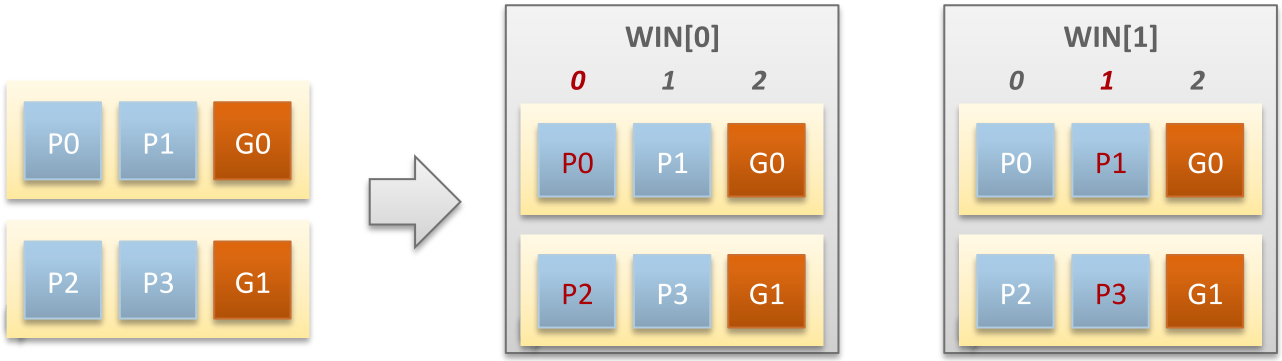
Lock (G0, win) Lock (G0, win) …

✗

MPI standard:
An origin cannot nest locks to the same target

Lock Permission Management for Shared Ghost Processes (2)

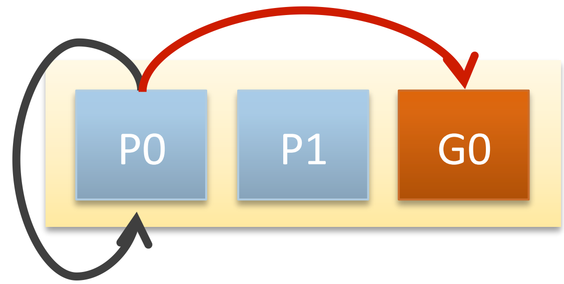
- **Solution**
  - **N Windows**
    - N = max number of processes on every node
    - COMM. to **i_{th} user process on each node** goes to **i_{th} window**



- **User hint optimization**
  - Window info "**epochs_used**" (fence|pscw|lock|lockall by default)
    - **If "epochs_used" contains "lock", create N windows**
    - Otherwise, only create a single window

Min Si msi@anl.gov
Argonne National Laboratory, The University of Tokyo

[Correctness Challenge 2] Self Lock Consistency (1)



P0

```
Lock (P0, win)

x=1
y=2
...

Unlock(P0, win)
```

MPI standard:
Local lock must be acquired immediately

→

```
Lock (G0, win)

Unlock(G0, win)
```

✗ MPI standard:
Remote lock may be delayed..

[Correctness Challenge 2] Self Lock Consistency (2)

- **Solution (2 steps)**
  1. **Force-lock with HIDDEN BYTES***

     ```
     Lock (G0, win)
     Get (G0, win)
     Flush (G0, win)      // Lock is acquired
     ```

  2. **Lock self**

     ```
     Lock (P0, win)       // memory barrier for managing
                          // memory consistency
     ```

- **User hint optimization**
  - Window info **no_local_loadstore**
    - Do not need both 2 steps
  - Epoch assert **MPI_MODE_NOCHECK**
    - Only need the 2_{nd} step

* MPI standard defines **unnecessary restriction on concurrent GET and accumulate**. See MPI Standard Version 3.0 , page page 456, line 39.

[Correctness Challenge 3]
Managing Multiple Ghost Processes (1)

1. **Lock permission among multiple ghost processes**

[INCORRECT] Two **EXCLUSIVE locks** to the same target may be **concurrently acquired**

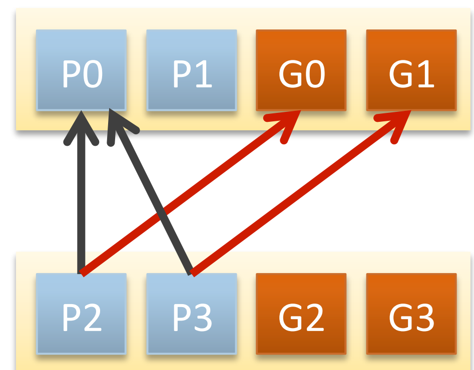
P2

```
Lock (EXCLUSIVE, P0, win)
PUT(P0)
Unlock(P0, win)
```

P3

Serialized

```
Lock (EXCLUSIVE, P0, win)
PUT(P0)
Unlock(P0, win)
```



P2

```
Lock (EXCLUSIVE, G0, win)
Lock (EXCLUSIVE, G1, win)

// get G0
G = randomly_pick_ghost();
PUT(G)
…
```

P3

```
Lock (EXCLUSIVE, G0, win)
Lock (EXCLUSIVE, G1, win)

// get G1
G = randomly_pick_ghost();
PUT(G)
…
```

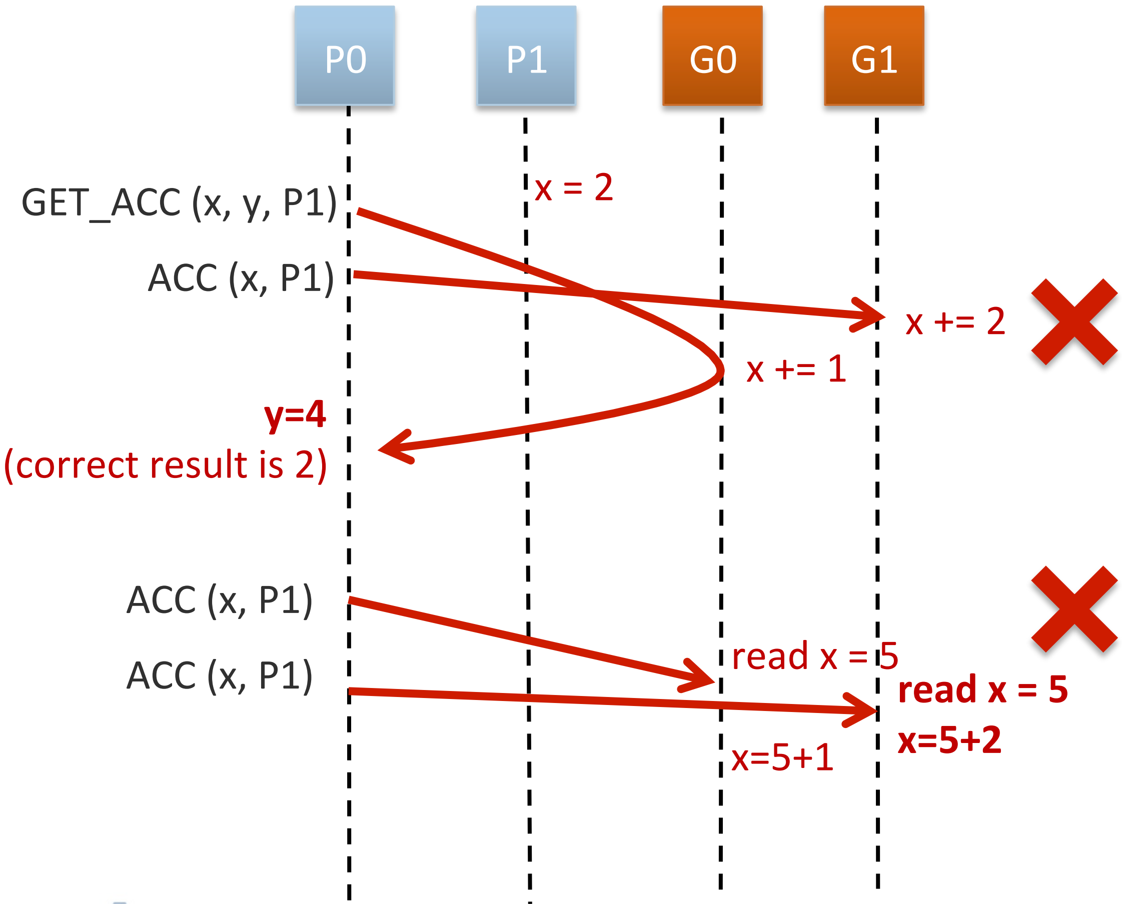
Empty lock can be ignored,
P2 and P3 may concurrently acquire lock on G0 and G1

[Correctness Challenge 3]

Managing Multiple Ghost Processes (2)

2. Ordering and Atomicity constraints for Accumulate operations

[INCORRECT] Ordering and Atomicity cannot be maintained by MPI among multiple ghost processes



MPI standard:
Same origin && same target location accumulates **must be ordered**

MPI standard:
Concurrent accumulates to the same target location **must be atomic per basic datatype element.**

[Correctness Challenge 3]

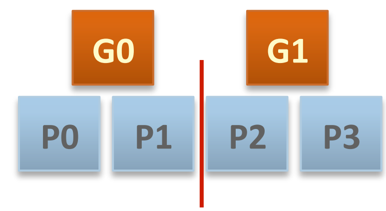
Managing Multiple Ghost Processes (3)

- **Solution (2 phases)**

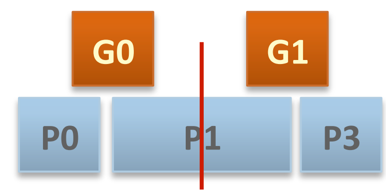
1. **Static-Binding Phase**
   - **Rank binding model**
     - **Each user process** binds to a single ghost process
   - **Segment binding model**
     - **Segment total exposed memory** on each node **into N_G chunks**
     - **Each chunk** binds to a single ghost process
   - Only redirect RMA operations to the bound ghost process
   - Fixed lock and ACC ordering & atomicity issues
   - But **only suitable for balanced communication patterns**

Optimization for dynamic communication patterns

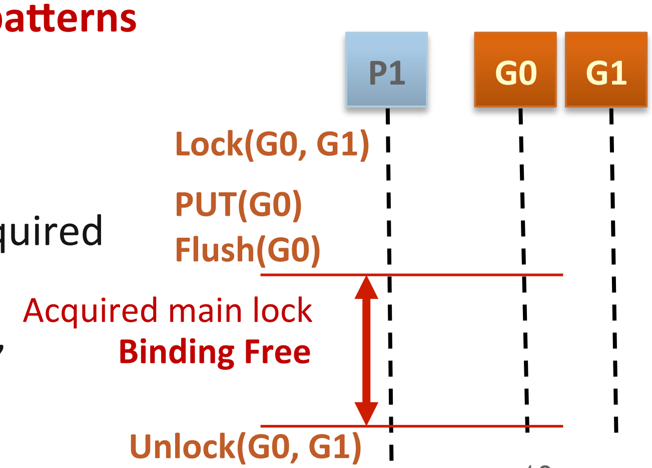
2. **Static-Binding-Free Phase**
   - **After operation + flush issued**, "main lock" is acquired
   - **Dynamically select target ghost process**
   - Accumulate operations can not be "binding free"



Static-rank-binding



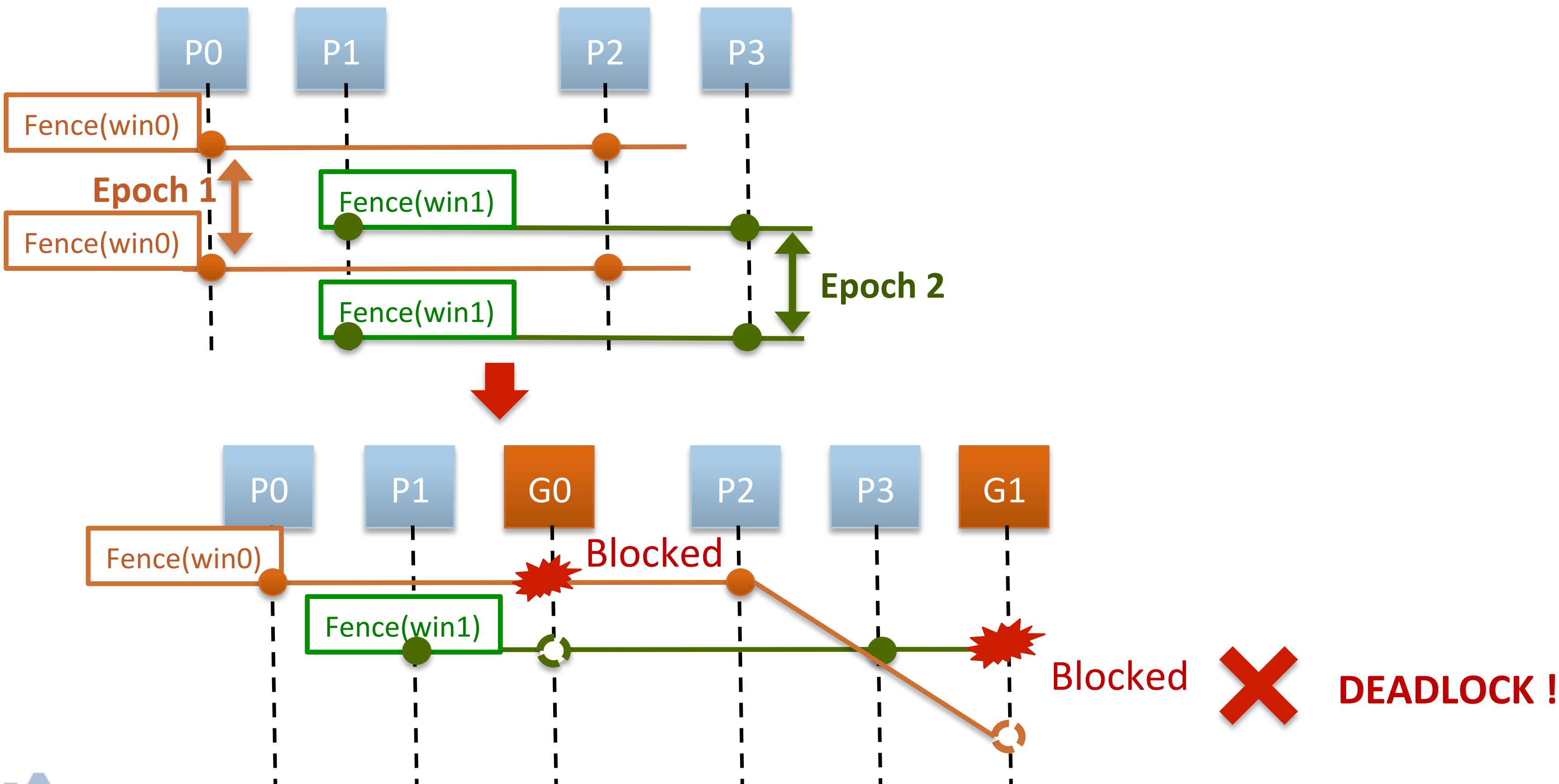
Static-segment-binding



[Correctness Challenge 4]
Multiple Simultaneous Epochs – Active Epochs (1)

- **Simultaneous fence epochs on disjoint sets of processes sharing the same ghost processes**

  [INCORRECT] Deadlock !



[Correctness Challenge 4]
Multiple Simultaneous Epochs - Active Epochs (2)

- **Solution**
  - Every user window has an **internal "global window"**
  - **Translate to passive-target mode**
  - **Fence**

Performed on user processes

User program	Performed on user processes
Win_allocate	Win_allocate **Lock_all (global win)**
Fence(win0) PUT(P) … **Fence(win)**	**Flush_all (global win) + Barrier(COMM_USER_WORLD) + Win_sync** PUT(G) … **Flush_all (global win) + Barrier(COMM_USER_WORLD) + Win_sync**
	[Performance issue 1] User hint **MPI_MODE_NOPRECEDE** avoids it **[Performance issue 2]** User hint **(NOSTORE & NOPUT & NOPRECEDE)** avoids it **[Performance issue 3]**
Win_free	**Unlock_all (global win)** Win_free

  - **PSCW → Flush + Send-Receive**

Evaluation

1. **Asynchronous Progress Microbenchmark**
2. **NWChem Quantum Chemistry Application**

Experimental Environment

- NERSC's newest supercomputer *
- Cray XC30

THE UNIVERSITY OF TOKYO

U.S. DEPARTMENT OF ENERGY

* https://www.nersc.gov/users/computational-systems/edison/configuration/

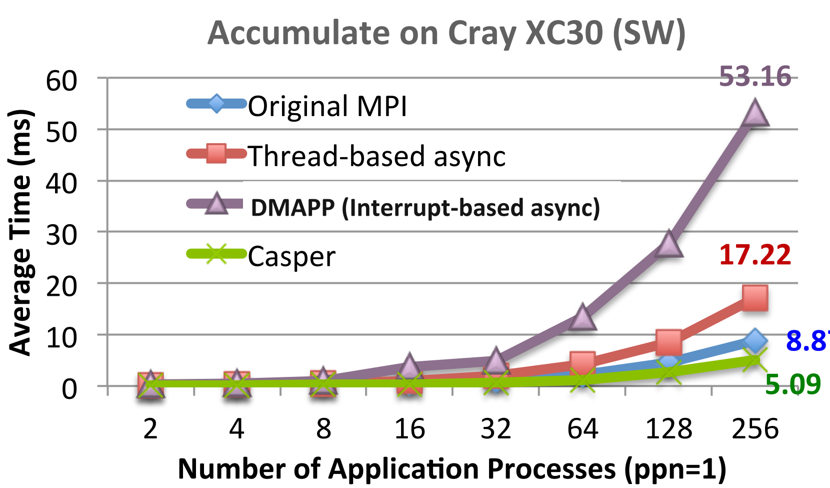
Asynchronous Progress Microbenchmark

RMA implementation in Cray MPI v6.3.1

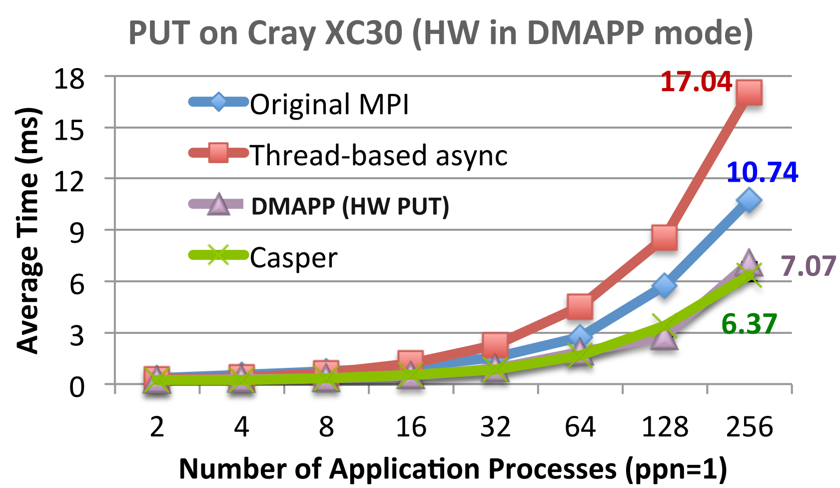
	HW-handled OP	ASYNC. mode
Original mode	NONE	Thread
DMAPP mode	Contig. PUT/GET	Interrupt

Test scenario

```
Lock_all (win);
for (dst=0; dst<nproc; dst++) {
        OP(dst, double, cnt = 1, win);
        Flush(dst, win);
        busy wait 100us; /*computing*/
}
Unlock_all (win)
```



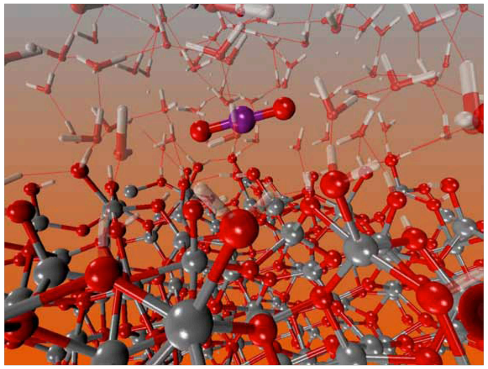
Casper provides asynchronous progress for SW-handled ACC.

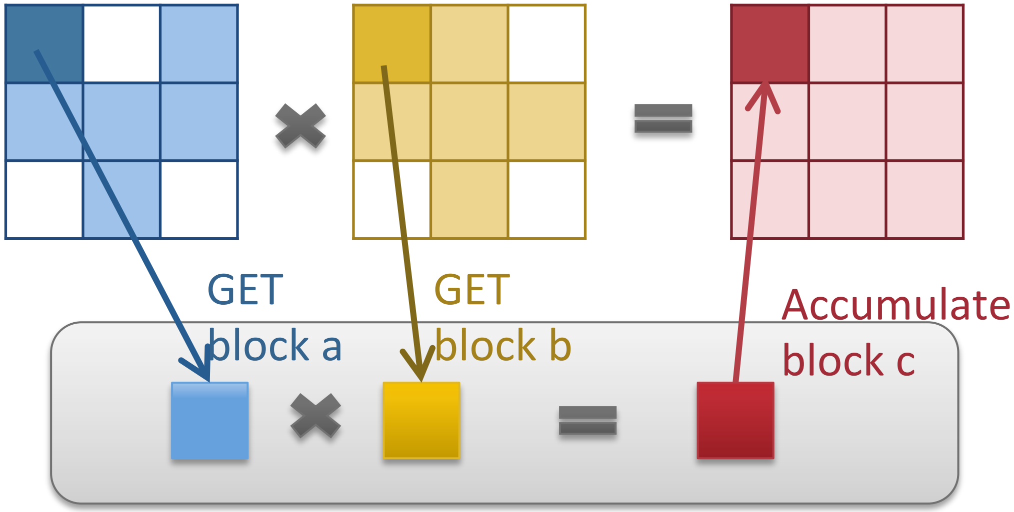


Casper does not affect the performance of HW PUT

NWChem Quantum Chemistry Application (1)

- Computational chemistry application suite composed of many types of simulation capabilities.
- **ARMCI-MPI** (Portable implementation of **Global Arrays over MPI RMA**)
- Focus on most common used **CC (coupled-cluster) simulations** in a C_{20} molecules





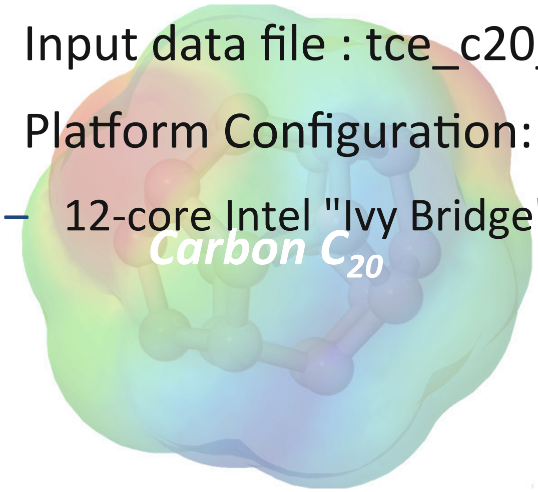
Perform DGEMM in local buffer

```
for i in I blocks:
  for j in J blocks:
    for k in K blocks:
      GET block a from A
      GET block b from B
      c += a * b /*computing*/
    end do
    ACC block c to C
  end do
end do
```

Get-Compute-Update model

Evaluation 2. NWChem Quantum Chemistry Application (2)

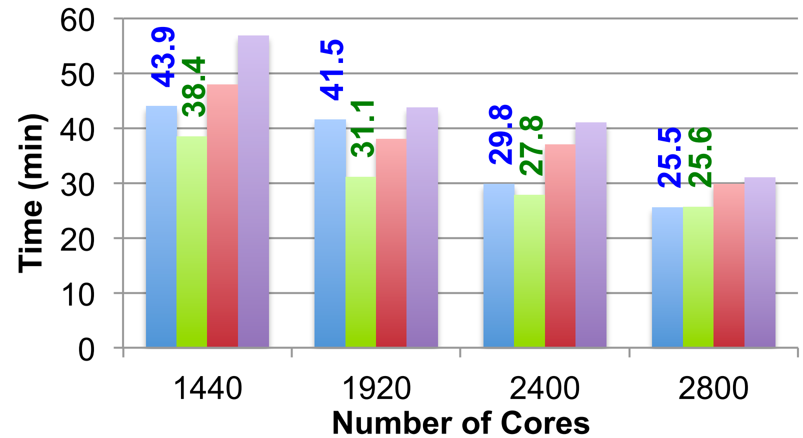
- Input data file : tce_c20_triplet
- Platform Configuration:
  - 12-core Intel "Ivy Bridge" (**24 cores per node**)



Core deployment

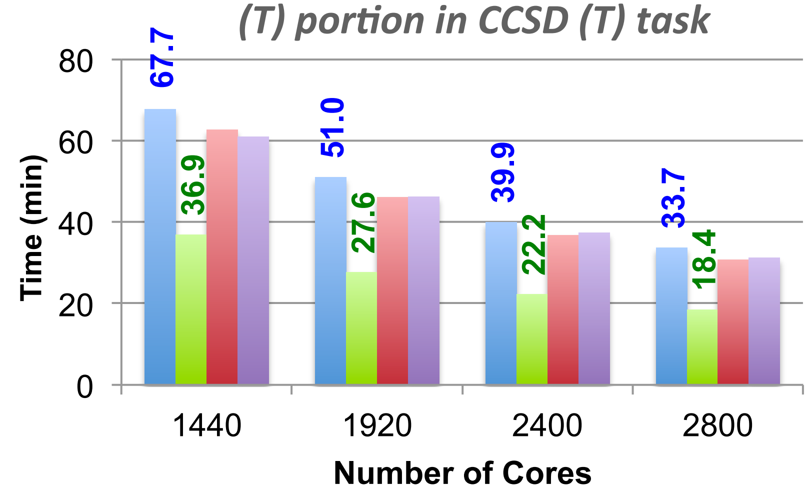
	# COMP.	# ASYNC.
Original MPI	24	0
Casper	20	4
Thread-ASYNC (oversubscribed)	24	24
Thread-ASYNC (dedicated)	12	12

CCSD iteration in CCSD task



Casper ASYNC. Progress helps CCSD performance

(T) portion in CCSD (T) task



More compute-intensive than CCSD, more improvement

Summary

- MPI RMA communication is **not truly one-sided**
  - Still **need asynchronous progress**
  - Additional overhead in thread / interrupt-based approaches
- Multi- / Many-Core architectures
  - Number of cores is growing rapidly, **some cores are not always busy**
- **Casper: a process-based asynchronous progress model**
  - **Dedicating arbitrary number of cores** to ghost processes
  - **Mapping window regions** from user processes to ghost processes
  - **Redirecting all RMA SYNC. & operations** to ghost processes
  - Linking to various MPI implementation through **PMPI transparent redirection**

Download slides: http://sudalab.is.s.u-tokyo.ac.jp/~msi/pdf/ipdps2015-casper-slides.pdf

Min Si msi@anl.gov
Argonne National Laboratory, The University of Tokyo