ICL
INNOVATIVE
COMPUTING LABORATORY

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

Plan B: Interruption of Ongoing MPI Operations to Support Failure Recovery

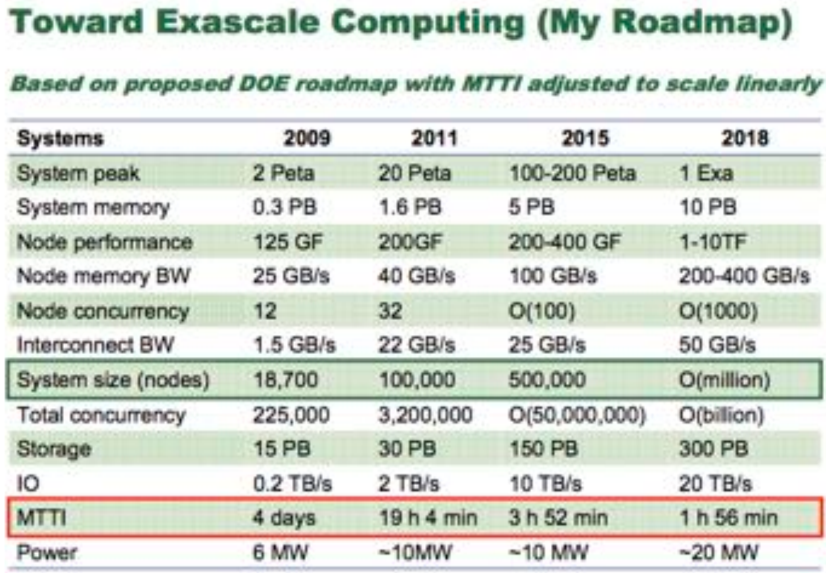
Aurelien Bouteiller

Scheduling Workshop
Nashville, May 19, 2016

Do we need fault tolerance?

Courtesy of
C. Engelman & S. Scott

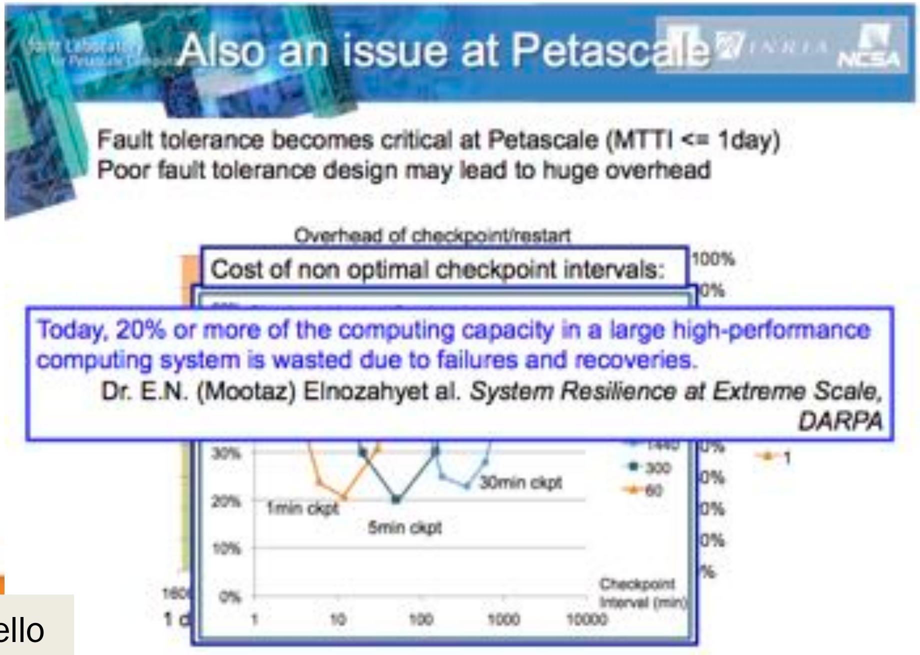
- No !
  - Hardware can take care of everything. And [of course] will !
    - The future tense is important !
    - At what cost ($, energy)?
- Meanwhile from a HPC viewpoint
  - Large platforms report several hard failures a day with tens/hundreds of applications to be rerun
  - ECC might not be enough to protect the data from Silent Data Corruptions
  - Future HPC platforms will grow in number of resources and by simple probabilistic deduction the frequency of faults will increase



Toward Exascale Computing (My Roadmap)

Based on proposed DOE roadmap with MTTI adjusted to scale linearly

Systems	2009	2011	2015	2018
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa
System memory	0.3 PB	1.6 PB	5 PB	10 PB
Node performance	125 GF	200GF	200-400 GF	1-10TF
Node memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	22 GB/s	25 GB/s	50 GB/s
System size (nodes)	18,700	100,000	500,000	O(million)
Total concurrency	225,000	3,200,000	O(50,000,000)	O(billion)
Storage	15 PB	30 PB	150 PB	300 PB
IO	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	4 days	19 h 4 min	3 h 52 min	1 h 56 min
Power	6 MW	~10MW	~10 MW	~20 MW



Also an issue at Petascale

Fault tolerance becomes critical at Petascale (MTTI <= 1day)
Poor fault tolerance design may lead to huge overhead

Overhead of checkpoint/restart

Cost of non optimal checkpoint intervals:

Today, 20% or more of the computing capacity in a large high-performance computing system is wasted due to failures and recoveries.
Dr. E.N. (Mootaz) Elnozahyet al. *System Resilience at Extreme Scale, DARPA*

Courtesy of F. Cappello

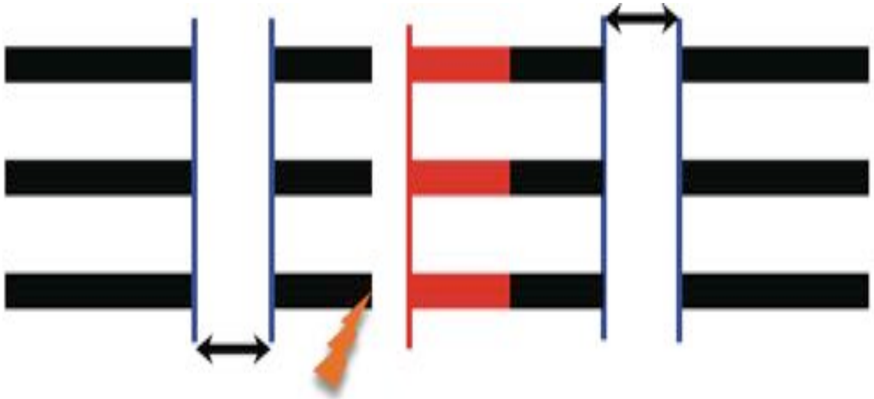
MPI-3: Fault Tolerance support

- We have algorithms (uncoordinated checkpoint, forward recovery), but they **expect MPI to continue to operate across failures**
  - MPI support of FT is non-existent
  - Prevents effective deployment of efficient, application specific approaches

  - MPI_ERRORS_ARE_FATAL (default mode)
    - Application crashes at first failure
  - MPI_ERRORS_RETURN
    - Error returned to the user
    - State of MPI ***undefined***
      - *"…does not necessarily allow the user to continue to use MPI after an error is detected. The purpose of these error handler is to allow a user to issue user-defined error messages and take actions unrelated to MPI…An MPI implementation is free to allow MPI to continue after an error…"* (MPI-1.1, page 195)
      - *"Advice to implementors: A good quality implementation will, to the greatest possible extent, circumvent the impact of an error, so that normal processing can continue after an error handler was invoked."*

Backward recovery: C/R

Coordinated checkpoint (possibly with incremental checkpoints)



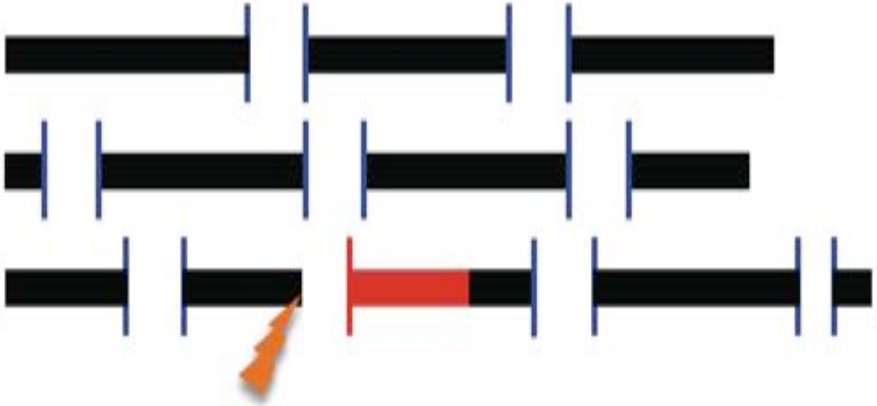
- Coordinated checkpoint is the workhorse of FT today
  - I/O intensive, significant failure free overhead ☹
  - Full rollback (1 fails, all rollback) ☹
  - Can be deployed w/o MPI support ☺

ULFM enables user-level deployment of in-memory, Buddy-checkpoints, Diskless checkpoint

  - Checkpoints stored on other compute nodes
  - No I/O activity (or greatly reduced), full network bandwidth

---

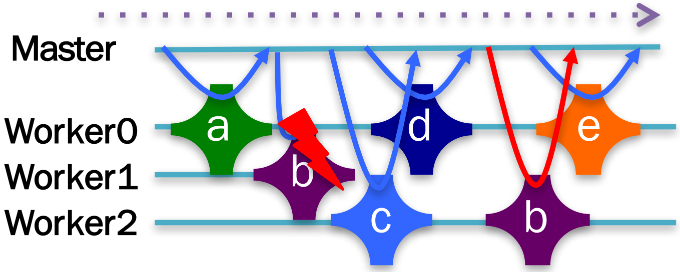
Uncoordinated checkpoint (message logging)

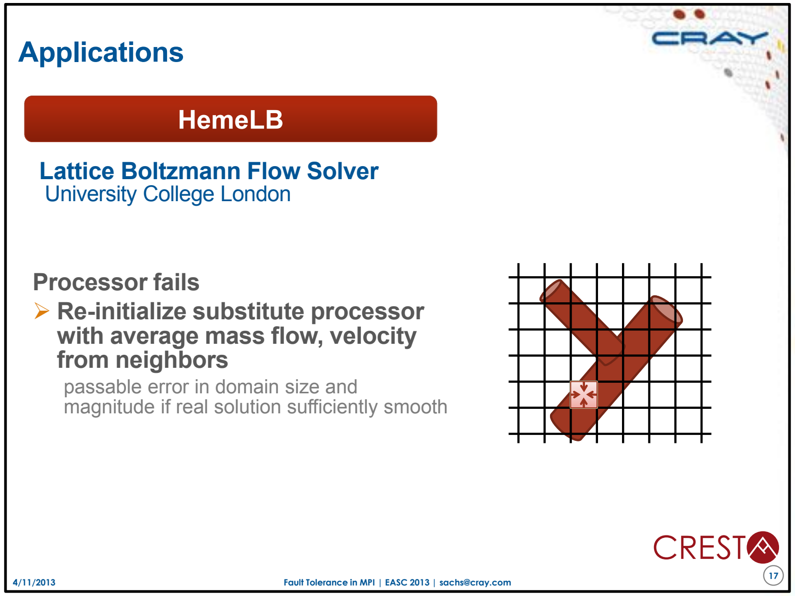


- Checkpoints taken independently
- Based on variants of Message Logging
- 1 fails, 1 rollback
- Can be implemented w/o a standardized user API
- Benefit from ULFM: implementation becomes portable across multiple MPI libraries

Forward Recovery

- Forward Recovery: *Any technique that permit the application to continue without rollback*
  - Master-Worker with simple resubmission
  - Iterative methods, Naturally fault tolerant algorithms
  - Algorithm Based Fault Tolerance
  - Replication *(the only system level Forward Recovery)*
- No checkpoint I/O overhead
- No rollback, no loss of completed work
- May require (sometime expensive, like replicates) protection/recovery operations, *but still generally more scalable than checkpoint* ☺
- Often requires in-depths algorithm rewrite (in contrast to automatic system based C/R) ☹





Applications

HemeLB

Lattice Boltzmann Flow Solver
University College London

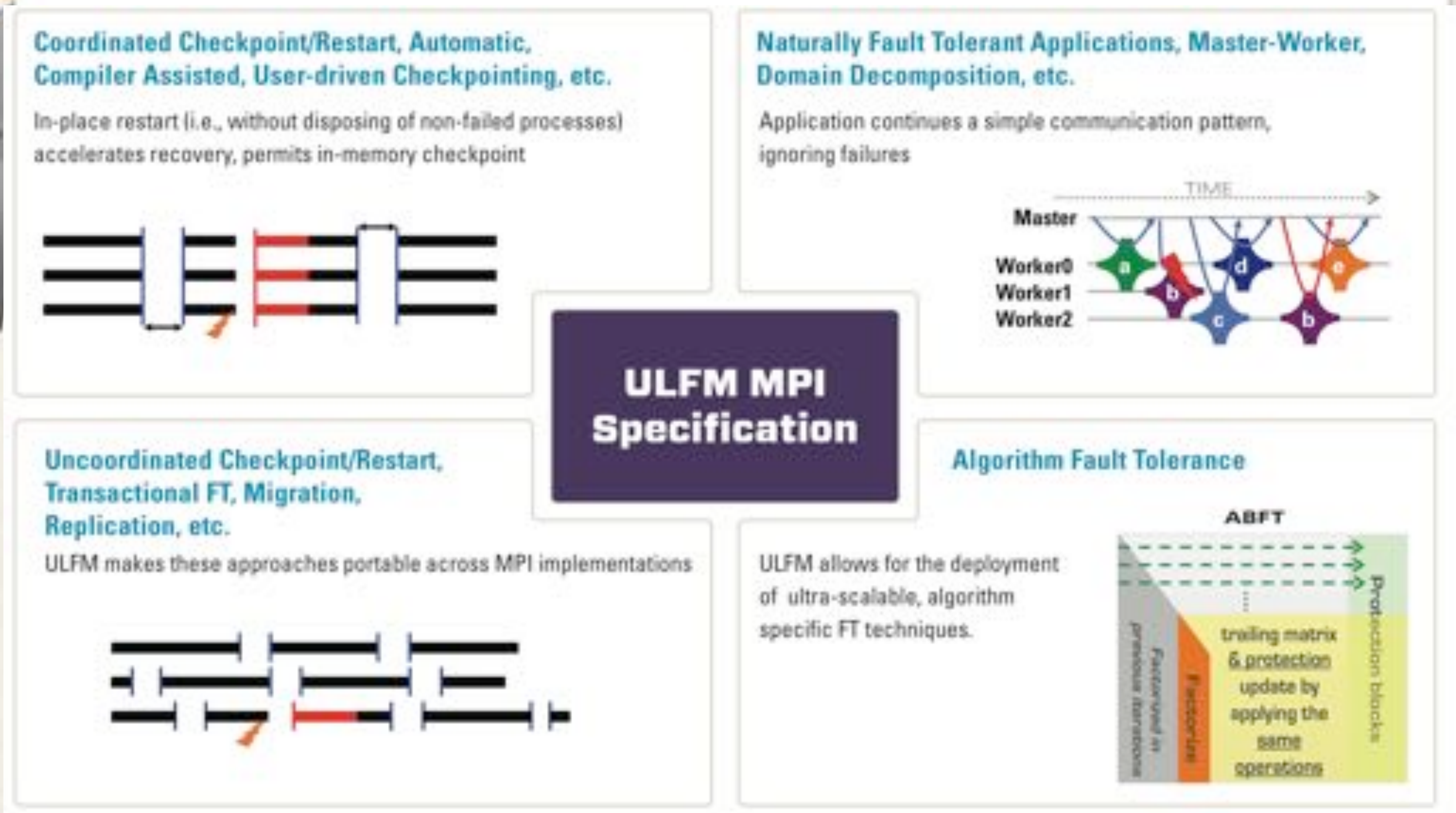
Processor fails

- **Re-initialize substitute processor with average mass flow, velocity from neighbors**
  passable error in domain size and magnitude if real solution sufficiently smooth

CRESTA

4/11/2013 Fault Tolerance in MPI | EASC 2013 | sachs@cray.com 17

Application Recovery Patterns



Coordinated Checkpoint/Restart, Automatic, Compiler Assisted, User-driven Checkpointing, etc.

In-place restart (i.e., without disposing of non-failed processes) accelerates recovery, permits in-memory checkpoint

Naturally Fault Tolerant Applications, Master-Worker, Domain Decomposition, etc.

Application continues a simple communication pattern, ignoring failures

ULFM MPI Specification

Uncoordinated Checkpoint/Restart, Transactional FT, Migration, Replication, etc.

ULFM makes these approaches portable across MPI implementations

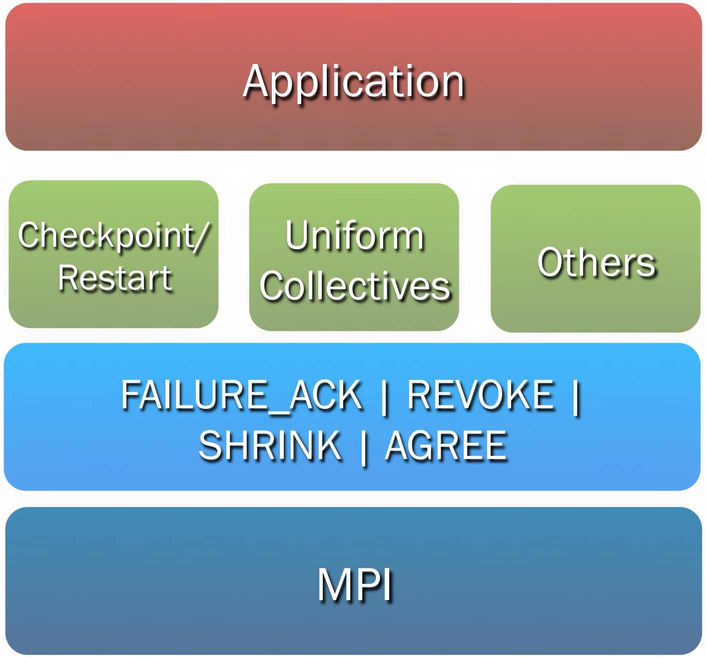
Algorithm Fault Tolerance

ULFM allows for the deployment of ultra-scalable, algorithm specific FT techniques.

User Level Failure Mitigation: a set of MPI interface extensions to enable MPI programs to restore MPI communication capabilities disabled by failures

Requirements for MPI standardization of FT

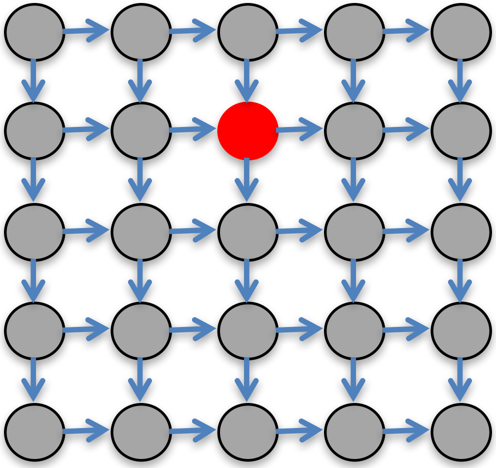
- Expressive, simple to use
  - Support legacy code, backward compatible
  - Enable users to port their code simply
  - Support a variety of FT models and approaches
- Minimal (ideally **zero) impact** on failure free performance
  - No global knowledge of failures
  - No supplementary communications to maintain global state
  - Realistic memory requirements
- Simple to implement
  - Minimal (or **zero) changes to existing functions**
  - Limited number of new functions
  - Consider thread safety when designing the API



Minimal Feature Set for a Resilient MPI

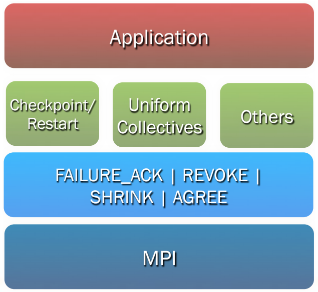
- Failure Notification
- Error Propagation
- Error Recovery

Not all recovery strategies require all of these features, that's why the interface splits notification, propagation and recovery.



What is the scope of a failure?
Who should be notified about?

ULFM is not a recovery strategy, but a minimalistic set of building blocks for more complex recovery strategies.



Application

Checkpoint/ Restart
Uniform Collectives
Others

FAILURE_ACK | REVOKE | SHRINK | AGREE

MPI

Integration with existing mechanisms

- New error codes to deal with failures
  - MPI_ERROR_PROC_FAILED: report that the operation discovered a newly dead process. Returned from all blocking function, and all completion functions.
  - MPI_ERROR_PROC_FAILED_PENDING: report that a non-blocking MPI_ANY_SOURCE potential sender has been discovered dead.
  - MPI_ERROR_REVOKED: a communicator has been declared improper for further communications. All future communications on this communicator will raise the same error code, with the exception of a handful of recovery functions

Summary of new functions

- MPI_Comm_failure_ack(comm)
  - Resumes matching for MPI_ANY_SOURCE
- MPI_Comm_failure_get_acked(comm, &group)
  - Returns to the user the group of processes acknowledged to have failed

Notification

- MPI_Comm_revoke(comm)
  - Non-collective collective, interrupts all operations on comm (future or active, at all ranks) by raising MPI_ERR_REVOKED

Propagation

- MPI_Comm_shrink(comm, &newcomm)
  - Collective, creates a new communicator without failed processes (identical at all ranks)
- MPI_Comm_agree(comm, &mask)
  - Collective, agrees on the AND value on binary mask, ignoring failed processes (reliable AllReduce), and the return core

Recovery

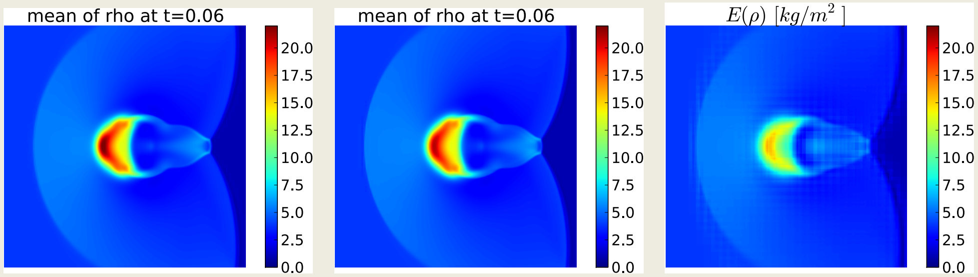
Bibliography of users’ activity

These works use ULFM

FRAMEWORKS USING ULFM

LFLR, FENIX, FTLA, Falanx, X10

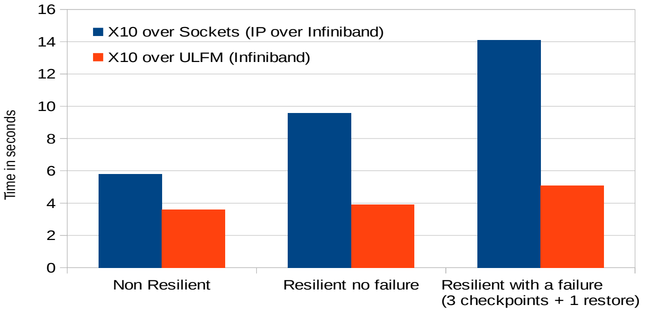
- HAMOUDA, Sara S., MILTHORPE, Josh, STRAZDINS, Peter E., *et al.* A Resilient Framework for Iterative Linear Algebra Applications in X10. In : *16th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2015)*. 2015.
- ST PAULI, P. Arbenz et SCHWAB, Ch. Intrinsic fault tolerance of multi level Monte Carlo methods. *ETH Zurich, Computer Science Department, Tech. Rep*, 2012.
- PAULI, Stefan, KOHLER, Manuel, et ARBENZ, Peter. A fault tolerant implementation of Multi-Level Monte Carlo methods. In : *PARCO*. 2013. p. 471-480.
- BLAND, Wesley, DU, Peng, BOUTEILLER, Aurelien, *et al.* Extending the scope of the Checkpoint-on-Failure protocol for forward recovery in standard MPI. *Concurrency and computation: Practice and experience*, 2013, vol. 25, no 17, p. 2381-2393.
- ALI, Md Mortuza, SOUTHERN, James, STRAZDINS, Peter, *et al.* Application Level Fault Recovery: Using Fault-Tolerant Open MPI in a PDE Solver. In : *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, 2014. p. 1169-1178.
- NAUGHTON, Thomas, ENGELMANN, Christian, VALLÉE, Geoffroy, *et al.*Supporting the development of resilient message passing applications using simulation. In : *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. IEEE, 2014. p. 271-278.
- ENGELMANN, Christian et NAUGHTON, Thomas. Improving the Performance of the Extreme-scale Simulator. In : *Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*. IEEE Computer Society, 2014. p. 198-207.
- TERANISHI, Keita et HEROUX, Michael A. Toward Local Failure Local Recovery Resilience Model using MPI-ULFM. In : *Proceedings of the 21st European MPI Users' Group Meeting*. ACM, 2014. p. 51.
- ALI, Md Mohsin, STRAZDINS, Peter E., HARDING, Brendan, *et al.* A fault-tolerant gyrokinetic plasma application using the sparse grid combination technique. In : *High Performance Computing & Simulation (HPCS), 2015 International Conference on*. IEEE, 2015. p. 499-507.
- VALLÉE, Geoffroy, NAUGHTON, Thomas, BOHM, Swen, *et al.* A runtime environment for supporting research in resilient HPC system software & tools. In : *Computing and Networking (CANDAR), 2013 First International Symposium on*. IEEE, 2013. p. 213-219.
- ZOUNMEVO, Judicael A., KIMPE, Dries, ROSS, Robert, *et al.* Extreme-scale computing services over MPI: Experiences, observations and features proposal for next-generation message passing interface. *International Journal of High Performance Computing Applications*, 2014, vol. 28, no 4, p. 435-449.
- NAUGHTON, Thomas, BÖHM, Swen, ENGELMANN, Christian, *et al.* Using Performance Tools to Support Experiments in HPC Resilience. In : *Euro-Par 2013: Parallel Processing Workshops*. Springer Berlin Heidelberg, 2014. p. 727-736.
- ENGELMANN, Christian et NAUGHTON, Thomas. A NETWORK CONTENTION MODEL FOR THE EXTREME-SCALE SIMULATOR.
- GAMELL, Marc, KATZ, Daniel S., KOLLA, Hemanth, *et al.* Exploring automatic, online failure recovery for scientific applications at extreme scales. In : *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014. p. 895-906.
- XIAOGUANG, Ren, XINHAI, Xu, YUHUA, Tang, *et al.* An Application-Level Synchronous Checkpoint-Recover Method for Parallel CFD Simulation. In : *Computational Science and Engineering (C*
- Judicael A. Zounmevo, Dries Kimpe, Robert Ross, and Ahmad Afsahi. 2013. Using MPI in high-performance computing services. In *Proceedings of the 20th European MPI Users' Group Meeting* (EuroMPI '13). ACM, New York, NY, USA, 43-48.*SE), 2013 IEEE 16th International Conference on*. IEEE, 2013. p. 58-65.
- Jinho Ahn, "N Fault-Tolerant Sender-Based Message Logging for Group Communication-Based Message Passing Systems," in *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on* , vol., no., pp.1296-1301, 19-21 Dec. 2014.



(a) failure-free (b) few failures (c) many failures

Credits: ETH Zurich

Figure 5. Results of the FT-MLMC implementation for three different failure scenarios.



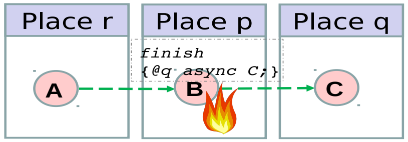
The performance improvement due to using ULFM v1.0 for running the LULESH proxy application [3] (a shock hydrodynamics stencil based simulation) running on 64 processes on 16 nodes with

User projects: Resilient X10

- X10 is a PGAS programming language
  - Legacy resilient X10 TCP based

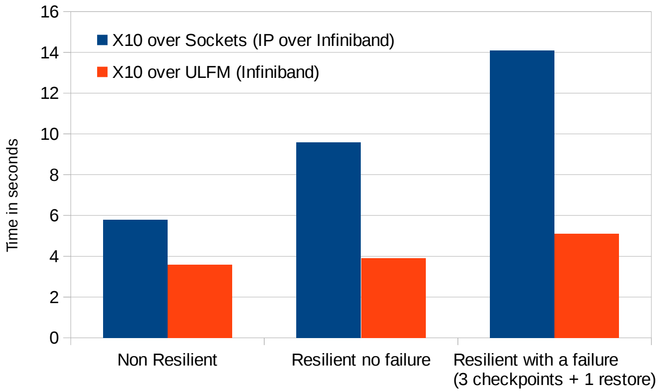
Happens Before Invariance Principle (HBI): *Failure of a place should not alter the happens before relationship between statements at the remaining places.*

```
try{ /*Task A*/
 at (p) { /*Task B*/
  finish { at (q) async { /*Task C*/ } }
 }
} catch(dpe:DeadPlaceException){ /*recovery steps*/}
D;
```



By applying the HBI principle, Resilient X10 will ensure that statement D executes after Task C finishes, despite the loss of the synchronization construct (finish) at place p

- MPI operations in resilient X10 runtime
  - Progress loop does MPI_Iprobe, post needed recv according to probes
  - Asynchronous background collective operations (on multiple different comms to form 2d grids, etc).
- Recovery
  - Upon failure, all communicators recreated (from shrinking a large communicator with spares, or using MPI_COMM_SPAWN to get new ones)
  - Ranks reassigned identically to rebuild the same X10 "teams"
- Injection of FT layer
  - Unnecessary, x10 has a runtime that hides all MPI from the application and handles failures internally

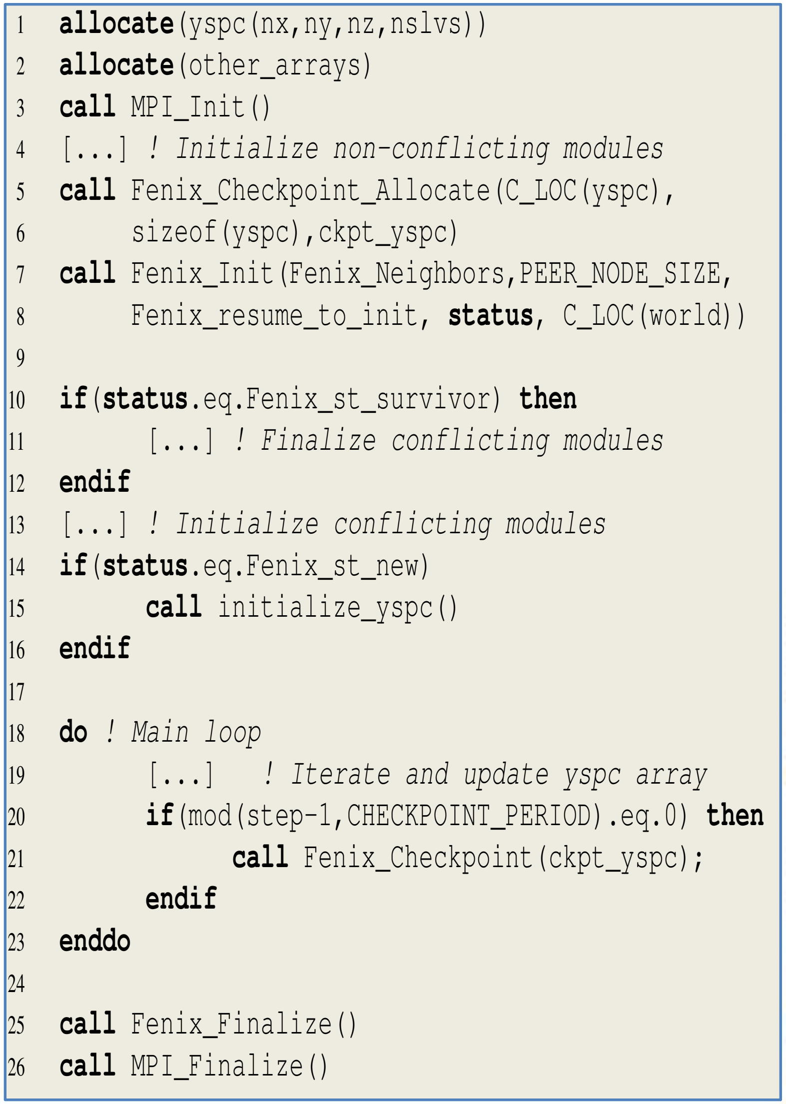


The performance improvement due to using ULFM v1.0 for running the LULESH proxy application [3] (a shock hydrodynamics stencil based simulation) running on 64 processes on 16 nodes with

Source: Sara Hamouda, Benjamin Herta, Josh Milthorpe, David Grove, Olivier Tardieu. *Resilient X10 over Fault Tolerant MPI.* In : poster session SC'15, Austin, TX, 2015.

User projects: Fenix+S3D

- Fenix is a framework to provide scoped user level checkpoint/restart
  - Provides some of the same services provided by the "MPI_Reinit" idea floated around by T. Gamblin
  - Recover failed processes with revoke-shrink-spawn-reoder sequence
  - Revovered and surviving processes jump back to the start (longjump in Fenix_init)
  - Fenix has helpers to perform user directed "in-memory" or "buddy" checkpointing (and reload)
  - Injection of FT layer: PMPI based
- **Fenix_Checkpoint_Allocate** mark a memory segment (baseptr,size) as part of the checkpoint.
- **Fenix_Init** Initialize Fenix, and restart point after a recovery, status contains info about the restart mode
- **Fenix_Comm_Add** can be used to notify Fenix about the creation of user communicators
- **Fenix_Checkpoint** performs a checkpoint of marked segments

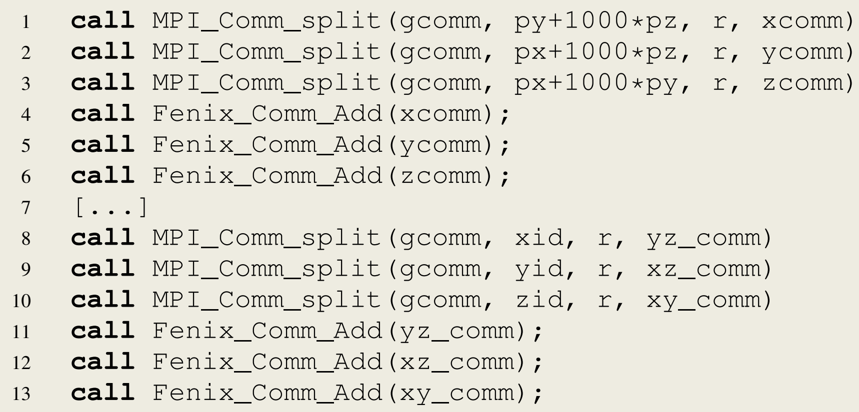


```
allocate(yspc(nx,ny,nz,nslvs))
allocate(other_arrays)
call MPI_Init()
[...] ! Initialize non-conflicting modules
call Fenix_Checkpoint_Allocate(C_LOC(yspc),
     sizeof(yspc),ckpt_yspc)
call Fenix_Init(Fenix_Neighbors,PEER_NODE_SIZE,
     Fenix_resume_to_init, status, C_LOC(world))

if(status.eq.Fenix_st_survivor) then
      [...] ! Finalize conflicting modules
endif
[...] ! Initialize conflicting modules
if(status.eq.Fenix_st_new)
      call initialize_yspc()
endif

do ! Main loop
      [...]   ! Iterate and update yspc array
      if(mod(step-1,CHECKPOINT_PERIOD).eq.0) then
            call Fenix_Checkpoint(ckpt_yspc);
      endif
enddo

call Fenix_Finalize()
call MPI_Finalize()
```

GAMELL, Marc, KATZ, Daniel S., KOLLA, Hemanth, *et al.* Exploring automatic, online failure recovery for scientific applications at extreme scales. In : *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014. p. 895-906.

User projects: Fenix+S3D

- S3D is a production, highly parallel method-of-lines solver for PDEs
  - used to perform first-principles-based direct numerical simulations of turbulent combustion
- S3D rendered fault tolerant using Fenix/ULFM
- 35 lines of code modified in S3D in total!
- Order of magnitude performance improvement in failure scenarios
  - thanks to online recovery and in-memory checkpoint advantage over I/O based checkpointing
- Injection of FT layer: addition of a couple of Fenix calls



```
call MPI_Comm_split(gcomm, py+1000*pz, r, xcomm)
call MPI_Comm_split(gcomm, px+1000*pz, r, ycomm)
call MPI_Comm_split(gcomm, px+1000*py, r, zcomm)
call Fenix_Comm_Add(xcomm);
call Fenix_Comm_Add(ycomm);
call Fenix_Comm_Add(zcomm);
[...]
call MPI_Comm_split(gcomm, xid, r, yz_comm)
call MPI_Comm_split(gcomm, yid, r, xz_comm)
call MPI_Comm_split(gcomm, zid, r, xy_comm)
call Fenix_Comm_Add(yz_comm);
call Fenix_Comm_Add(xz_comm);
call Fenix_Comm_Add(xy_comm);
```

S3D Code snippet to declare to Fenix the communicators to recover

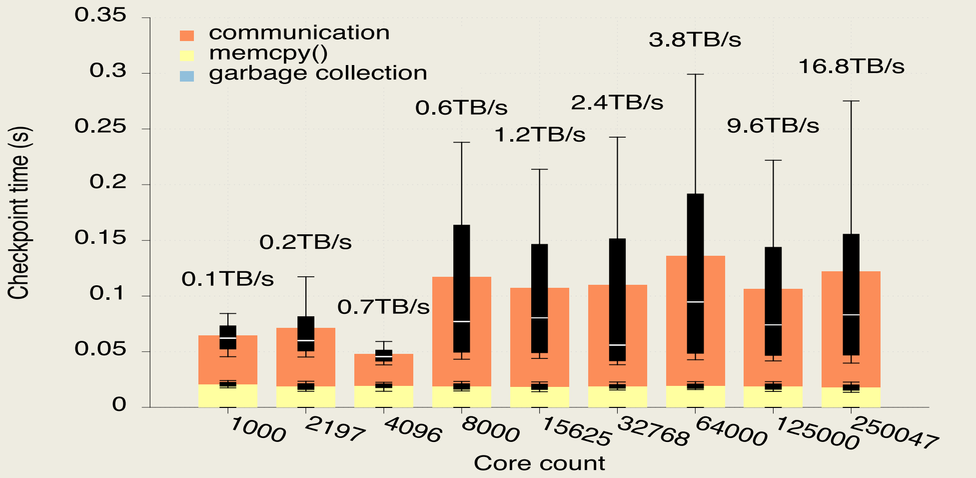


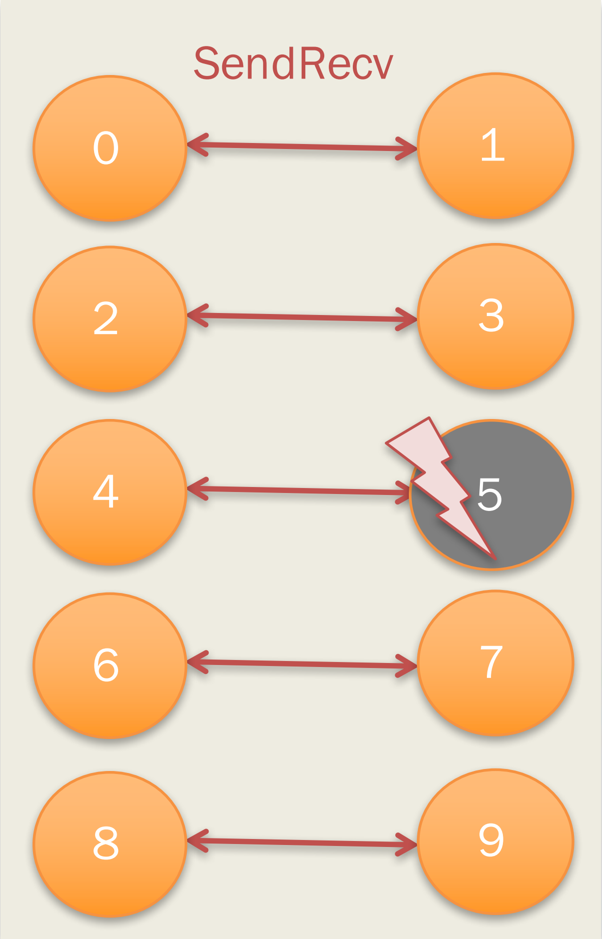
Fig. 3. Checkpoint time for different core counts (8.6 MB/core). The numbers above each test show the aggregated bandwidth (the total checkpoint size over the average checkpoint time).

Pictures and examples from M. Gamel&al SC14' paper

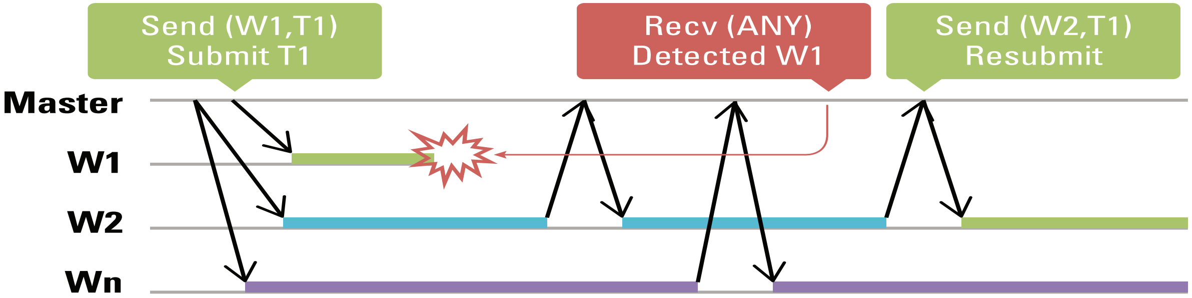
Errors are visible only for operations that can't complete

- Operations that can't complete return ERR_PROC_FAILED
  - State of MPI objects unchanged (communicators, etc)
  - Repeating the same operation has the same outcome
- Operations that can be completed return MPI_SUCCESS
  - Pt-2-pt operations between non failed ranks can continue

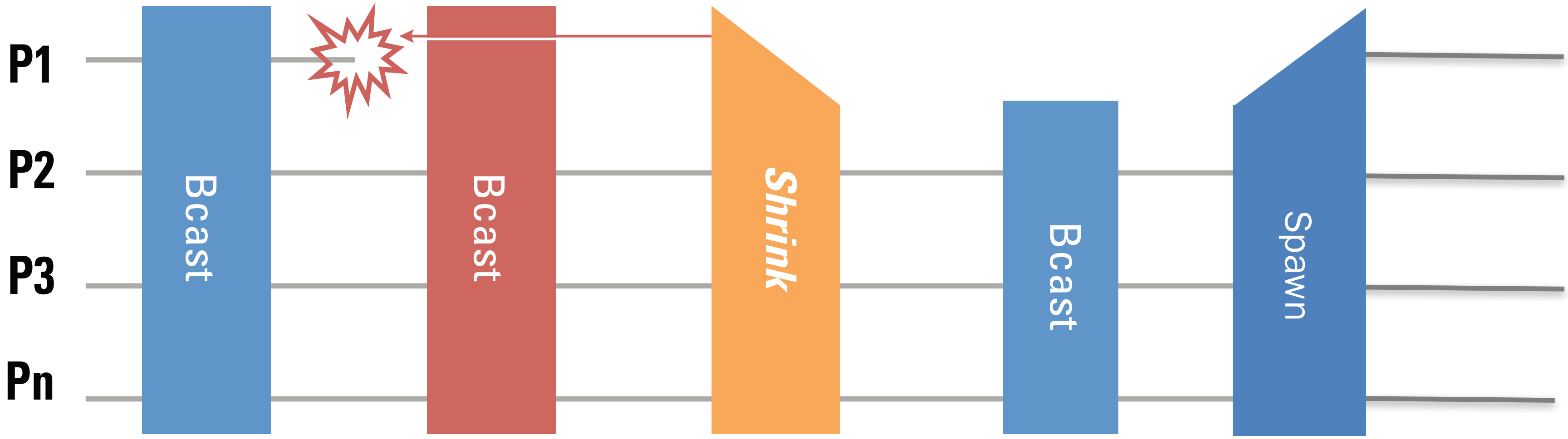
Example: only rank4 should report the failure of rank 5



This model is enough to support M/W etc.

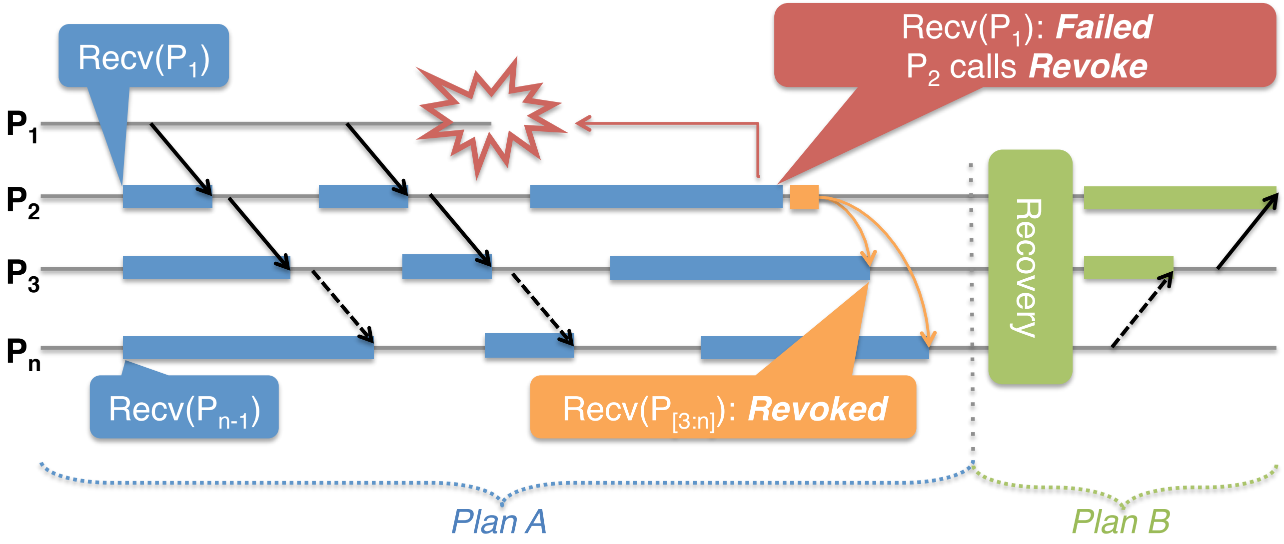


Full Capabilities Recovery



- Some applications are moldable
  - Shrink creates a new communicator on which collectives work
- Some applications are not moldable
  - Spawn can recreate a “same size” communicator
  - It is easy to reorder the ranks according to the original ordering

Resolving transitive dependencies



```
proc_failed_err_handler(MPI_Comm comm, int err) {
   if(err == MPI_ERR_PROC_FAILED ||
      err == MPI_ERR_REVOKED ) {
     if(err == MPI_ERR_PROC_FAILED) MPI_Comm_revoke(comm);
     recovery(comm);
   }
}
ft_transitive_deps(void)  {
   for(i=0; i<nbrecv; i++) {
      if(myrank>0) MPI_Irecv(buff, count, datatype,
                                    myrank-1, tag, comm, &req);
      if(myrank<n) MPI_Send(buff2, count, datatype,
                                    myrank+1, tag, comm, &req); }
}
```

- P1 fails
  - P2 raises an error and wants to change comm pattern to do application recovery
  - but P3..Pn are stuck in their posted recv
  - P2 can unlock them with Revoke
  - P3..Pn join P2 in the recovery

Errors and Collective Communications

```
proc_failed_err_handler(MPI_Comm comm, int err) {
    if(err == MPI_ERR_PROC_FAILED ||
        err == MPI_ERR_REVOKED ) {
       if(err == MPI_ERR_PROC_FAILED) MPI_Comm_revoke(comm);
       recovery(comm);
    }
}
deadlocking_collectives(void) {
    for(i=0; i<nbrecv; i++)
        MPI_Bcast(buff, count, datatype, 0, comm);
}
```

- Lax consistency: Exceptions are raised only at ranks where the Bcast couldn't succeed
    - In a tree-based Bcast, only the subtree under the failed process sees the failure
    - Other ranks succeed and proceed to the next Bcast
    - Ranks that couldn't complete enter "recovery", do not match the Bcast posted at other ranks => MPI_Comm_revoke(comm) interrupts unmatched Bcast and forces an exception (and triggers recovery) at all ranks

Revoke is a critical operation that must be reliable and scalable

Contribution 1:
MPI_Comm_revoke != Reliable Broadcast

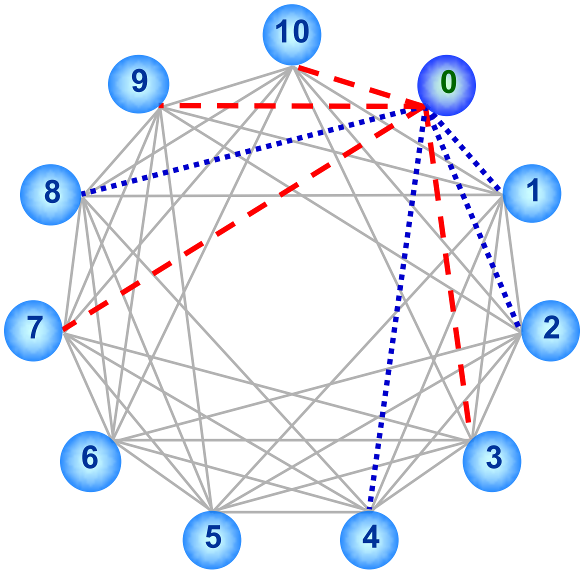
- The revoke notification need to be propagated to all alive processes (almost like a reliable broadcast)
- In the context of MPI_Comm_revoke, the 4 defining qualities of a reliable broadcast (Termination, Validity, Integrity and Agreement) can be relaxed (non-uniform versions)
  - Agreement, Validity: once one process delivers *v*, then all processes delivers *v*. Revoke has a single state (revoked) and all processes will eventually converge their views.
  - Integrity: a message delivered at most once. The revoked communicator is immutable, so multiple deliveries is not an issue
  - Termination: Once a communicator is locally known as revoked no further propagation of the state change
- As we don't need uniform variants of the revoke operation, we are not bound to fully-connected overlay topologies (Hamiltonian is more than enough)

Contribution 2: Identifying a suitable underlying topology

- The basic behavior of a process: once it receives a revoke message for the first time it delivers it to all neighbors
  - The agreement property can only be guaranteed when failures do not disconnect the overlay graph
- Fully connected topologies do have such a property but they scale poorly with the number of processes. In practice:
  - Number of messages quadratic
  - Resource exhaustion: too many simultaneously opened channels, too many unexpected messages or posted receives
- We need a better topology with small degree and diameter, hardened and bridgeless
  - Torus, HiC, CST, Hypercube, Chord (not good enough)

Binomial Graph (BMG)

- Undirected graph $G:=(V, E)$, $|V|=n$ (any size)
  - Node $i=\{0,1,2,\ldots,n-1\}$ has links to a set of nodes U
    - $U=\{i\pm1, i\pm2, \ldots, i\pm2^k \mid 2^k \leq n\}$ in a circular space
    - $U=\{ (i+1) \bmod n, (i+2) \bmod n, \ldots, (i+2^k) \bmod n \mid 2^k \leq n \}$ and
      $\{ (n+i-1) \bmod n, (n+i-2) \bmod n, \ldots, (n+i-2^k) \bmod n \mid 2^k \leq n \}$

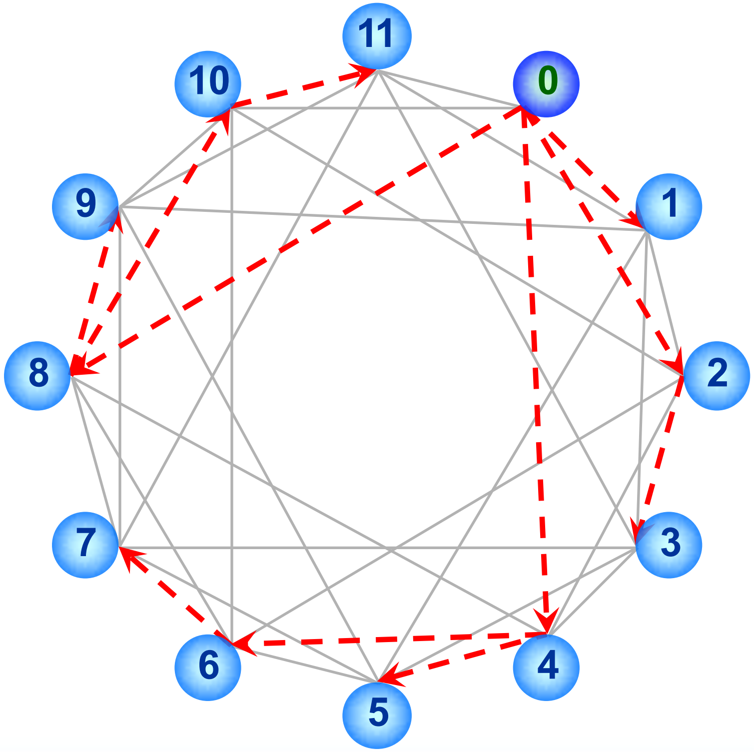


Belong to the connected Circulant graph family: biconnected, bridgeless, cyclic, Hamiltonian, LCF, regular, traceable, and vertex-transitive.

Angskun, T., Bosilca, G., Dongarra, J. "Binomial Graph: A Scalable and Fault-Tolerant Logical Network Topology," Proceedings of The Fifth International Symposium on Parallel and Distributed Processing and Applications (ISPA07), Springer, Niagara Falls, Canada, 2007

Binomial Graph (BMG)

- Merging all necessary links creates a binomial tree from each node in the graph.



Properties

1. Broadcast messages from any node within $\lceil \log_2(n) \rceil$ steps
2. Extremely difficult to bipartite
3. Easy to compute an alternate routing around failed processes
4. Interesting self-healing properties

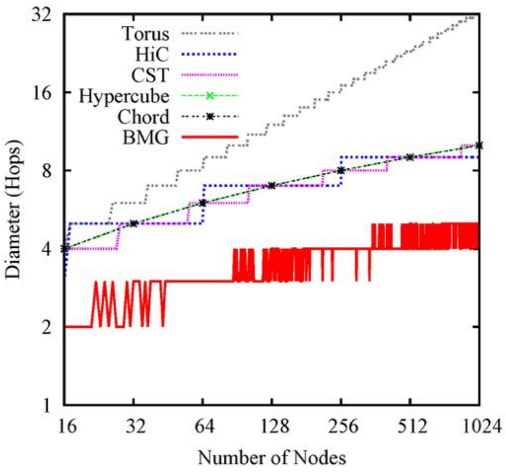
Basic Properties of BMG

- Degree δ (number of neighbors)

$$\delta = \begin{cases} (2 \times \lceil \log_2 n \rceil) - 1 & \text{For } n = 2^k\text{, where } k \in \mathbb{N} \\ (2 \times \lceil \log_2 n \rceil) - 2 & \text{For } n = 2^k + 2^j\text{, where } k, j \in \mathbb{N} \wedge k \neq j \\ 2 \times \lceil \log_2 n \rceil & \text{Otherwise} \end{cases}$$

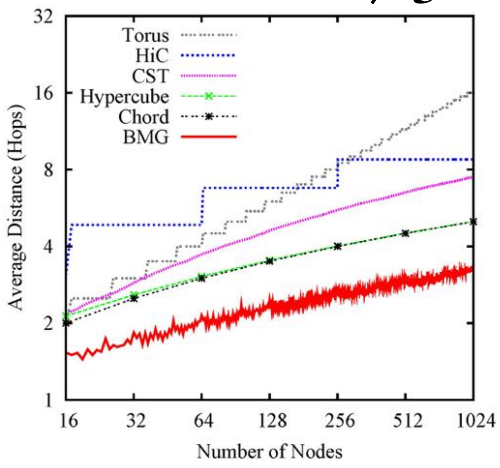
Diameter

$$(D) = O\left(\left\lceil \frac{\lceil \log_2(n) \rceil}{2} \right\rceil\right)$$

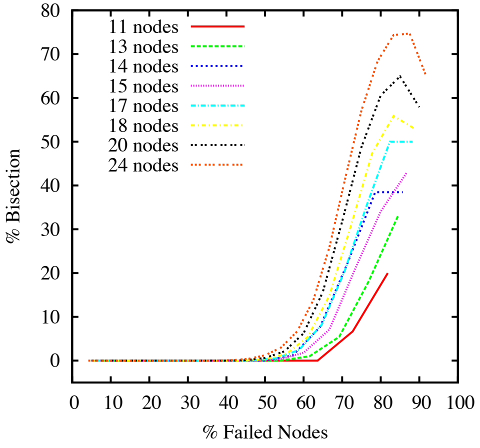


Average Distance

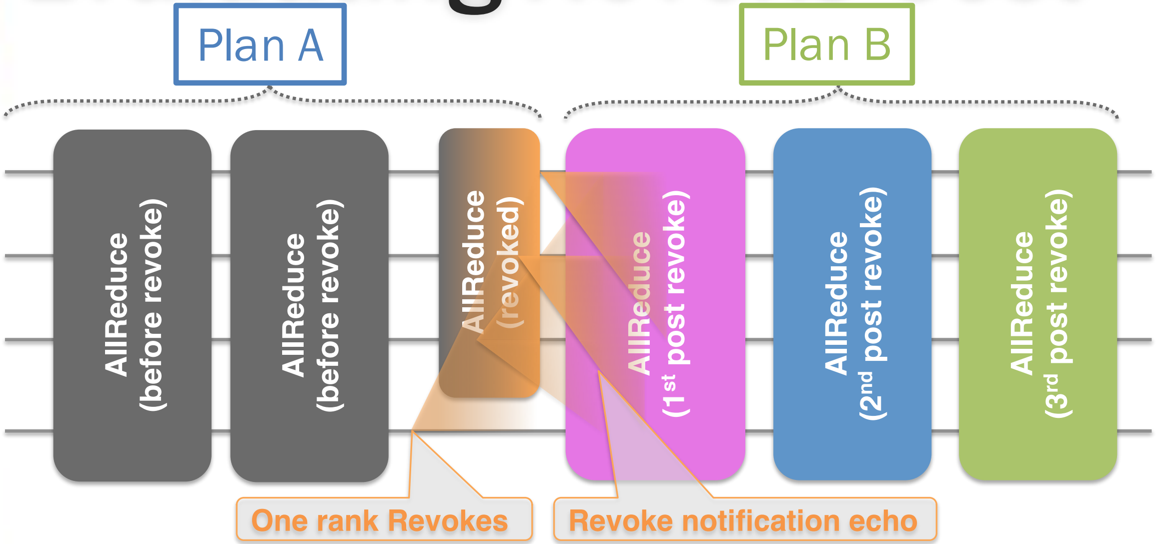
$$(\bar{d}) \approx {}^{\log_2(n)}\big/_{3}$$



Bipartite vs. Failed relationship



Evaluating Revoke Cost

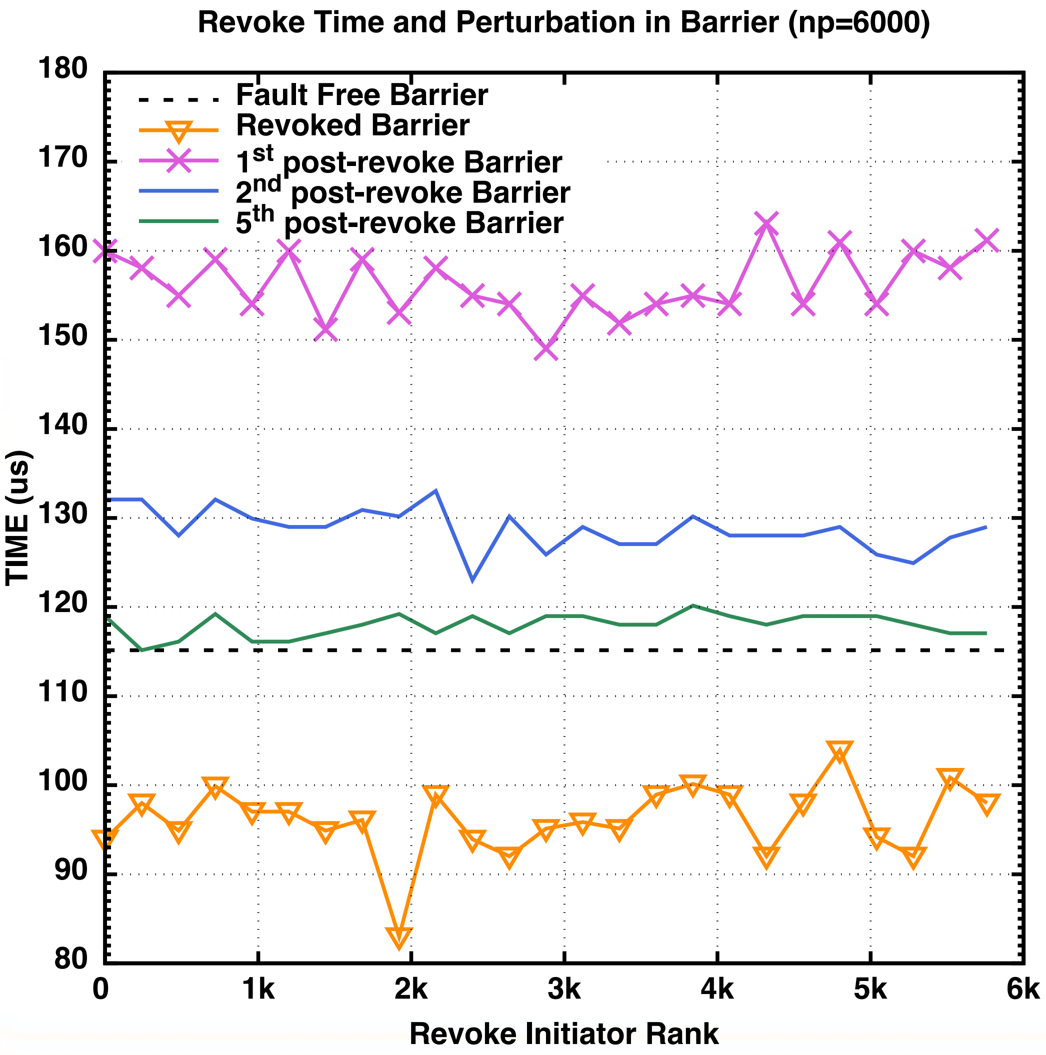


- The cost of Revoke cannot be measured directly. At the initial caller is essentially 0 (immediate operation, completes in the background)
- Instead we measure the impact of a revoke on subsequent operations
- Even after a Revoke has delivered to all ranks, the "revoke tokens" are still circulating on the network

- Two duplicate of MPI_COMM_WORLD:
- On the blue communicator:
  - Repeat allreduce (measure baseline time)
  - At some iteration, one rank revokes the blue communicator
  - Measure the time it takes for the last allreduce to be revoked at all ranks
- Immediately after, on the green communicator
  - Repeat allreduce (this comm is not revoked, no deads, so everything works w/o errors)
  - Measure the time it takes for the first, second, … collective, until the background noise generated by revoke cannot be observed

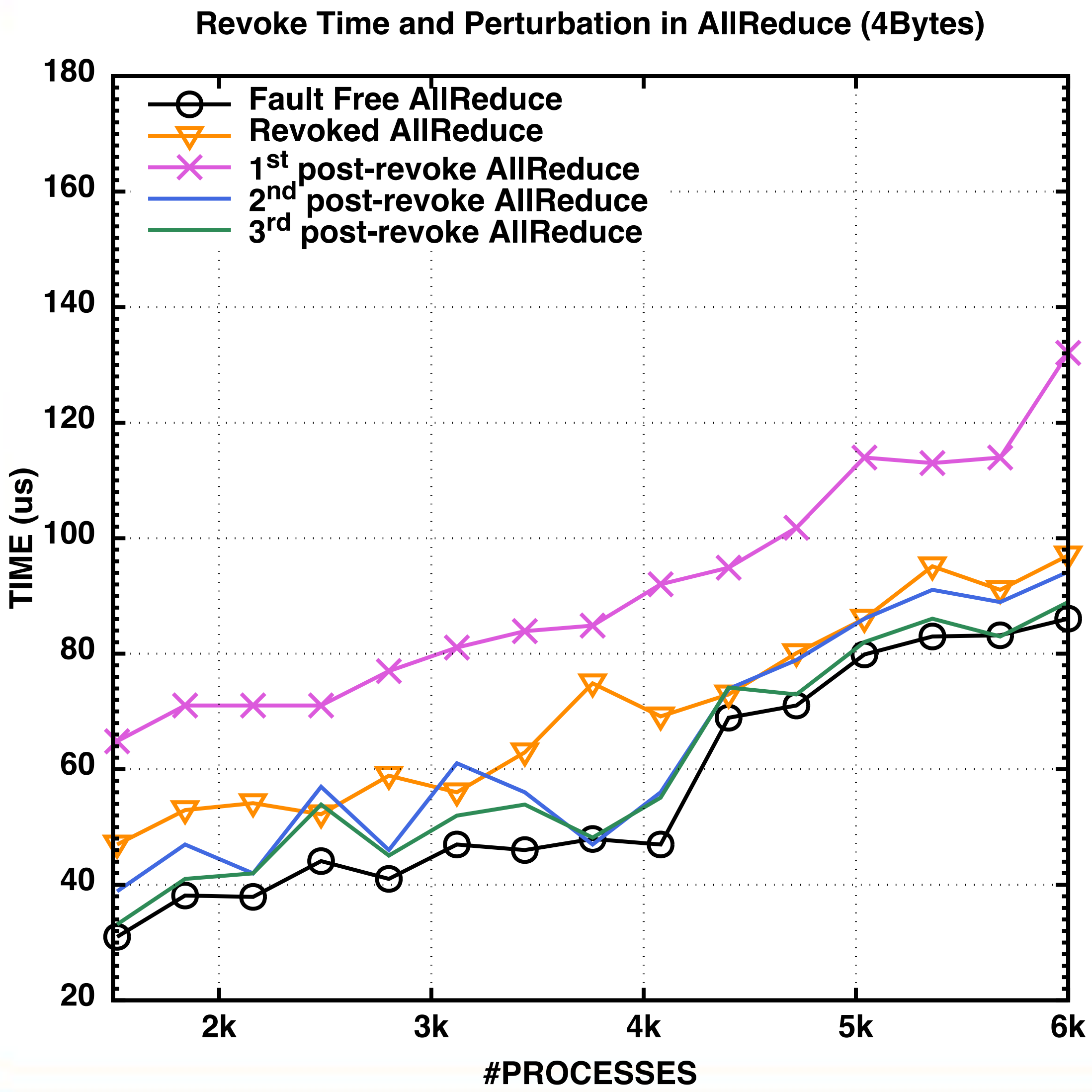
Darter platform, a Cray XC30 at NICS724 compute nodes with 2 x 2.6 GHz Intel 8-core XEON E5-2600 (Sandy Bridge), connected via a Cray Aries router with a bandwidth of 8GB/sec.

Evaluation: Initiator Location



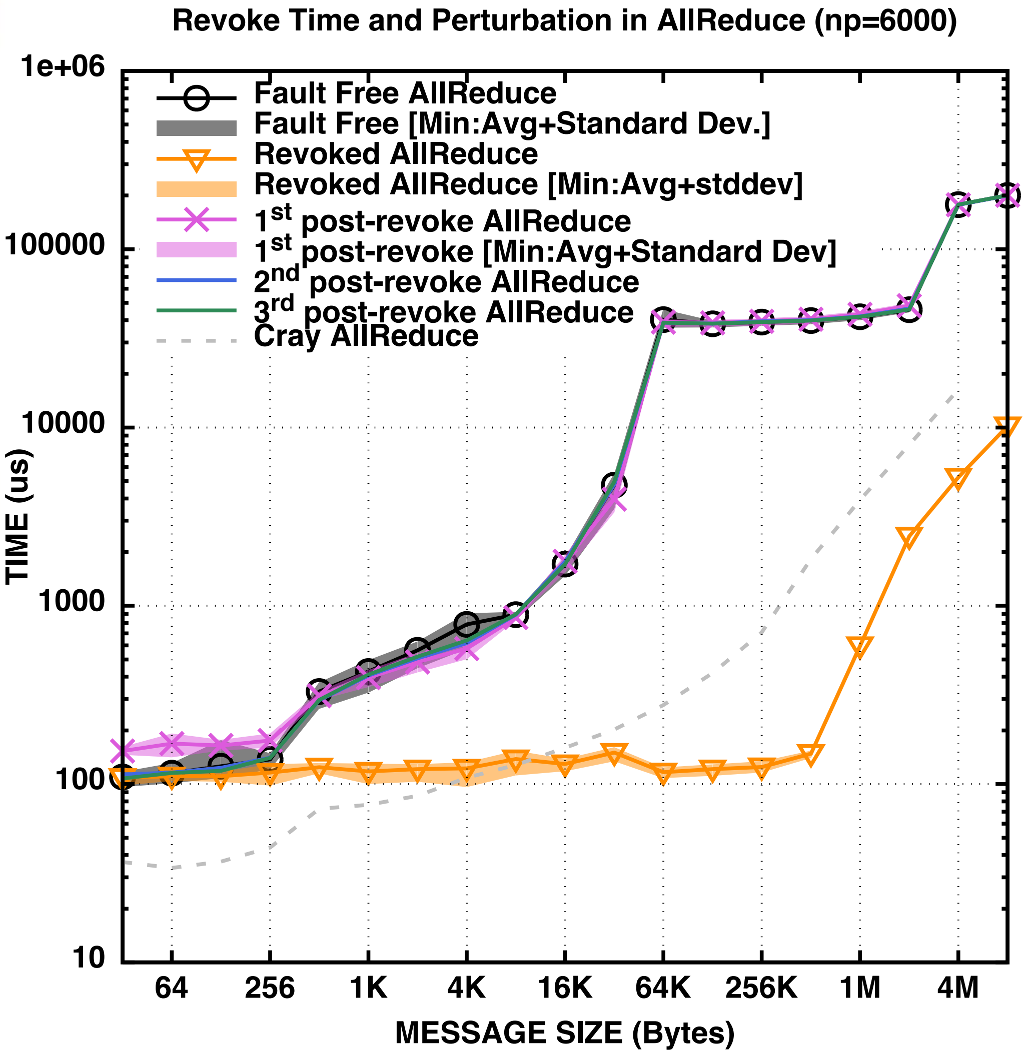
- The underlying BMG topology is symmetric and reflects in the revoke which is independent of the initiator
- The performance of the first post-Revoke collective operation sustains some performance degradation resulting from the network jitter associated with the circulation of revoke tokens
- After the fifth Barrier (approximately 700μs), the application is fully resynchronized, and the Revoke reliable broadcast has completely terminated, therefore leaving the application free from observable jitter.

Evaluation: Collective pattern



Performance of post-Revoke collective communications follows the same scalability trend as the pre-Revoke operations, even those impacted by jitter.

Evaluation: Message Size



- Propagation time for Revoke messages ~= small message allreduce latency
- After the revoke has propagated, noise continue for another small message allreduce latency
- Performance penalty only visible for small message operations and only for a short duration.

Conclusion

- ULFM is not a fault management approach
  - It's a toolbox to build higher-level application/domain specific techniques
  - Critical to improve the scalability and performance of the ULFM constructs
    - detection / revoke / agreement*
- There are now viable alternatives to handling the faults by C/R
  - HPC applications can definitively benefit
  - This makes MPI a suitable programming environment for domains outside HPC
- Scalable fault tolerant algorithmic building blocks
  - Applications beyond MPI (OpenSHMEM, runtime systems, etc).

* Herault, T., Bouteiller, A., Bosilca, G., Gamell, M., Teranishi, K., Parashar, M., Dongarra, J. "Practical Scalable Consensus for Pseudo-Synchronous Distributed Systems," SuperComputing, Austin, TX, November, 2015

More info, resources

http://fault-tolerance.org/

- Standard draft working group
  - https://github.com/mpiwg-ft/ft-issues/issues
- Prototype implementation available
  - Version 1.1 based on Open MPI 1.7 released late November 2015
    https://bitbucket.org/icldistcomp/ulfm
  - Full communicator-based (point-to-point and all flavors of collectives) support
  - Network support IB, uGNI, TCP, SM
  - RMA, I/O in progress