Undisrupted Applications

What we need from a message passing library make our applications resilient.

George Bosilca, UTK

Aurelien Bouteiller, Thomas Herault, Pierre Lemarinier, Wesley Bland, Graham Fagg, Thara Angskun, Jack Dongarra, Yulu Zia, Chawn Cao, Julien Langou, Yves Robert, The FTWG

PDSEC’16, Chicago, IL
Do we need fault tolerance?

- Optimist: Hardware can (and [of course] will) take care of everything.
  - Don’t we like future tense!
  - At what cost ($, energy)?

- Pessimist (Realist)
  - Large platforms report several hard failures a day with tangible impact on applications
  - Future HPC platforms will grow in number of resources and by simple probabilistic deduction the frequency of faults will increase
  - ECC might not be enough to protect the data from Silent Data Corruptions ...

<table>
<thead>
<tr>
<th>System attributes</th>
<th>2009</th>
<th>“Pre-Exascale”</th>
<th>“Exascale”</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 PF</td>
<td>100-200 PF/s</td>
<td>1 Exaflop/s</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>15 MW</td>
<td>20 MW</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>5 PB</td>
<td>32-64 PB</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>150 PB</td>
<td>500 PB</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>0.5 TF</td>
<td>7 TF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>0.1 TB/s</td>
<td>1 TB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(100)</td>
<td>O(1,000)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>500,000</td>
<td>50,000</td>
</tr>
<tr>
<td>Node interconnect BW</td>
<td>1.5 GB/s</td>
<td>150 GB/s</td>
<td>1 TB/s</td>
</tr>
<tr>
<td>IO Bandwidth</td>
<td>0.2 TB/s</td>
<td>10 TB/s</td>
<td>30-60 TB/s</td>
</tr>
<tr>
<td>MTTI</td>
<td>day</td>
<td>O(1 day)</td>
<td>O(0.1 day)</td>
</tr>
</tbody>
</table>
What are we protecting against?

• What fault are concerning: transient, intermittent, permanent, fail-stop, Byzantine.
  • Jaguar (360TB of memory) was logging 350 ECC errors per minute, and one unrecoverable memory corruption per day.
  • At Exascale we expect 64PB of memory, 180 times more than Jaguar. Transistors will shrink to 1/3 of today’s size, thus only 60 times more are subject to cosmic radiation.
  • That accounts for 60 unrecoverable memory corruptions per day, 2.5 per hour.
    • This little computation ignore the voltage reduction expected in order to met the power cap.

• Few tentative to provide a resilient parallel programming model
  • Notable efforts: PVM, CoArray, X10
  • MPI standardization body failed to converge to a resilience compromise because ...

IEEE Spectrum, March 2016
Fault Tolerance: many solutions

- **Rollback Recovery**
  - Not only legacy approach
  - Checkpoint/Restart based
  - Still advancing and improving

- **Forward Recovery**
  - Replication *(the only system level Forward Recovery)*
  - Master-Worker with simple resubmission
  - Iterative methods, Naturally fault tolerant algorithms
  - Algorithm Based Fault Tolerance

- No checkpoint, no message logging
- Small overhead due to synchronizations between replicas
- Benefit: Possible soft error detection and recovery (for deterministic applications)

---

**Figure 1:** Modeled application efficiency with and without state machine replication for a 168-hour application, 5-year per-socket MTBF, and 15 minute checkpoint times. Shaded region corresponds to possible socket counts for an exascale class machine.

*Evaluating the Viability of Process Replication Reliability for Exascale Systems*  
– Kurt Ferreira, Jon Stearley, James H. Laros III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Patrick G. Bridges, Dorian Arnold and Rolf Riesen – SC’11
Fault Tolerance: many solutions

- **Rollback Recovery**
  - Not only legacy approach
  - Checkpoint/Restart based
  - Still advancing and improving

- **Forward Recovery**
  - Replication *(the only system level Forward Recovery)*
  - Master-Worker with simple resubmission
  - Iterative methods, Naturally fault tolerant algorithms
  - Algorithm Based Fault Tolerance

Prediction for an Exascale machine
Memory per component remains constant
Problem size increases *(O(\sqrt{n}): matrix based)*
\( \mu \) at \( n=10^5 \) 1 day is \( O(1/n) \)
\( C (=R) \) at \( n=10^5 \) is 1m, is \( O(n) \)
Fault Tolerance: many solutions

- **Rollback Recovery**
  - Not only legacy approach
  - Checkpoint/Restart based
  - Still advancing and improving

- **Forward Recovery**
  - Replication (*the only system level Forward Recovery*)
  - Master-Worker with simple resubmission
  - Iterative methods, Naturally fault tolerant algorithms
  - Algorithm Based Fault Tolerance

- Hardware support (NVRAM)
- Decrease checkpoint size (*)
- Harden the checkpoint data

---

Prediction for a ≠ Exascale machine
Memory per component remains constant
Problem size increases \( O(\sqrt{n}) \): matrix based
\( \mu \) at \( n=10^5 \) 1 day is \( O(1/n) \)
\( C (=R) \) at \( n=10^5 \) is 1m, is in \( O(1) \)
Fault Tolerance: many solutions

- **Rollback Recovery**
  - Not only legacy approach
  - Checkpoint/Restart based
  - Still advancing and improving

- **Forward Recovery**
  - Replication (the only system level Forward Recovery)
  - Master-Worker with simple resubmission
  - Iterative methods, Naturally fault tolerant algorithms
  - Algorithm Based Fault Tolerance

- Any technique that permit the application to continue without rollback
  - 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

“Why is not everybody doing this already, then?”
Fault Tolerance: many solutions

- **Rollback Recovery**
  - Not only legacy approach
  - Checkpoint/Restart based
  - Still advancing and improving

- **Forward Recovery**
  - Replication (*the only system level Forward Recovery*)
  - Master-Worker with simple resubmission
  - Iterative methods, Naturally fault tolerant algorithms
  - Algorithm Based Fault Tolerance

- **Algorithm specific FT methods**
  - Very scalable, low overhead 😊
  - Not General, but...
  - These algorithms are part of many important applications
  - *Can’t be deployed w/o a fault tolerant Message Passing*
Minimal Feature Set for a Resilient parallel programming paradigm

1. Failure Notification
2. Error Propagation
3. Error Recovery

Other desirable features

1. Expressive: enable simple recovery with minimal intrusion, backward compatible (do nothing gives the same behavior as today), support thread safety and support a variety of fault management models
2. Minimal (ideally zero) impact on failure free performance: no global knowledge about failure when not needed
3. Simple to use: No alteration of existing API, limited number of new functions, and flexible
How do we take advantage of such capabilities

• In which programming model? Let’s assume MPI
  • Many original solutions have been proposed some evolving around these concepts and some orthogonal
  • MPI is about generic and portable and performance and productivity, so whatever solution we go for should satisfy as many of these MPI goals as possible.

• My take: User Level Fault Mitigation (ULFM)
  • Originally it was a particular implementation of a fault tolerant extension to MPI
  • Nowadays, it is also the name of the additional chapter releases by the FT working group in the MPI Forum working toward a standardization
    • Under evaluation for MPI 4.0
    • Reading in 2 weeks @ MPI Forum in Seattle
  • Get in touch with your representative!
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

- Existing mechanisms find a new purpose
  - MPI_COMM_SET_ERRHANDLER to capture and manage the new survivable error codes
  - MPI_COMM_SPAWN to restart replacement processes
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

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

- **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
Failure Notification

• For scalable message passing it would be unreasonable to expect full connectivity between all peers
• The failure detection and notification should have a neighboring scope: only processes involved in a communication with the failed process might detect the failure
• Failures Reported by an operation only when the operation cannot complete normally
Scalable Failure Detector

\(f = \text{supported number of overlapping failures}\)

Stabilization Time \(T(f) = \text{duration of the longest sequence of non stable configurations assuming at most } f \text{ overlapping faults}\)

Broadcast Time \(B(n) = 8\tau \log n\)

\[T(f) \leq f(f + 1)\delta + f\tau + \frac{f(f + 1)}{2}B(n)\]

The broadcast algorithm can tolerate up to \(\lfloor \log(n) \rfloor\) overlapping failures, thus

\[T(f) \sim O((\log n)^3)\]
Error Propagation

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

• Flexibility! Allow the programming model or library/application to decide the scope of a failure, and to limit the scope of a failure only to the concerned participants

  • eg. What is the difference between a Master/Worker and a tightly coupled application?
  • In a 2d mesh application how many nodes should be informed about a failure?
Scalable Resilient Error Propagation

The error propagation need to reach all alive processes (almost like a reliable broadcast)

But (in this context) 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

- BMG* Revoke propagation in less than 100µs
- 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 Revoke reliable broadcast has completely terminated, therefore leaving the application free from observable jitter.

Scalable Resilient Agreement

ERA Topologies (Cray XC30)

- Traditional Consensus choose a single value out of all proposed values
  - PAXOS
- We need more flexible construct that returns a value based on all proposed values by alive processes
  - The consensus does not have to be uniform
- Early Returning Agreement algorithm*
- Architecture-aware topology, scalable
- Only 2x the Cray AllReduce latency at 6k processors!

Algorithm-Based Fault Tolerance (ABFT)

• KH Huang & Jacob Abraham, ABFT for Matrix Operations, IEEE Trans. Computers. 01/1984;
  • Implementation on systolic arrays

• Takes advantage of additional mathematical relationship(s)
  • Already present in algorithm (GMRES)
  • Introduced (cheaply, if possible) by ABFT
ABFT Idea

- C matrix contains a checksum (row summations) of A
- Checksums introduce redundancy (resilience) to the algorithm and remain mathematically invariant!

\[ C_i = \sum_j A_{ij} \]

The Same algorithm updates both the trailing matrix AND the checksums

\[ \text{Update}(\sum_j A_{ij}) = \sum_j \text{Update}(A_{ij}) \]
ABFT Idea (validation)

- C matrix contains a checksum (row summations) of A
- Checksums introduce redundancy (resilience) to the algorithm and remain mathematically invariant!

\[ C_i = \sum_j A_{ij} \]

\[ e = (1, 1, \ldots, 1)^T \]

\[ Ae = ?c \]

Check the results after completion
ABFT Idea (recovery)

• C matrix contains a checksum (row summations) of A
• Checksums introduce redundancy (resilience) to the algorithm and remain mathematically invariant!

\[ C_i = \sum_j A_{ij} \]

\[ A_{ij} = C_i - \sum_{k \neq j} A_{ik} \]

In case of failure, checksum inversion allows to restore the missing value.
ABFT Overheads

Matrix $M \times N$, Blocks $mb \times nb$, Process grid $p \times q$

**Memory Overhead**

$$O\left(\frac{F}{q} \times M \times N\right)$$

Matrix is extended with 2F columns every q columns

**Computation Overhead**

$$O\left(\frac{F}{q} \times M^3\right)$$

flops for the checksum update, and

$$O(MN)$$

flops for the checksum creation.

*Less than 5% computational overhead*

N.B. Usually $F << q$

Relative overheads in $F/q$

e.g. 2 simultaneous faults on 192x192 process grid => 1% memory overhead

F: maximum number of *simultaneous* failures tolerated
ABFT Overheads

Matrix $M \times N$, Blocks $mb \times nb$, Process grid $p \times q$

**Memory Overhead**

$$O\left(\frac{F}{q} \times M \times N\right)$$

Matrix is extended with $2F$ columns every $q$ columns

**Computation Overhead**

$$O\left(\frac{F}{q} \times M^3\right)$$

flops for the checksum update, and

$$O(MN)$$

flops for the checksum creation.

Less than 5% computational overhead

N.B. Usually $F \ll q$

Relative overheads in $F/q$
e.g. 2 simultaneous faults on $192 \times 192$ process grid => 1% memory overhead

F: maximum number of *simultaneous* failures tolerated
Performance for QR

![Graph showing Performance (TFlop/s) vs Relative Overhead over ScaLAPACK (%). The x-axis represents #Processors (PxQ grid); Matrix size (N), with values including 6x6; 20k, 12x12; 40k, 24x24; 80k, and 48x48; 160k. The y-axis represents Performance (TFlop/s) and Relative Overhead over ScaLAPACK (%), with values ranging from 0 to 60.]
Performance for QR

![Performance Graph](image)

- ScaLAPACK PDGEQRF
- FT-PDGEQRF (no errors)

Overhead: FT-PDGEQRF (no errors)

Performance (TFlop/s) vs. Relative Overhead over ScaLAPACK (%)

- #Processors (PxQ grid); Matrix size (N)
Performance for QR

- ScaLAPACK PDGEQRF
- FT-PDGEQRF (no errors)
- Overhead: FT-PDGEQRF (no errors)
- FT-PDGEQRF (one error)
- Overhead: FT-PDGEQRF (one error)
Performance for LU

![Graph showing performance for LU]

- **Performance (TFlop/s)**
- **Relative Overhead over ScaLAPACK (%)**

- **#Processors (PxQ grid); Matrix size (N)**
  - 8x8; 62k
  - 16x16; 125k
  - 32x32; 250k
  - 64x64; 500k
  - 128x128; 1000k
Performance for LU

<table>
<thead>
<tr>
<th>#Processors (PxQ grid); Matrix size (N)</th>
<th>Performance (TFlop/s)</th>
<th>Relative Overhead over ScaLAPACK (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8x8; 62k</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16x16; 125k</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>32x32; 250k</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>64x64; 500k</td>
<td>500k</td>
<td>500k</td>
</tr>
<tr>
<td>128x128; 1000k</td>
<td>1000k</td>
<td>1000k</td>
</tr>
</tbody>
</table>

ScaLAPACK PDGESV

FT-PDGESV (no errors)

Overhead: FT-PDGESV (no errors)
Performance for LU

Performance (TFlop/s) vs. Relative Overhead over ScaLAPACK (%)

ScaLAPACK PDGESV
FT-PDGESV (no errors)
Overhead: FT-PDGESV (no errors)
FT-PDGESV (2 errors)
Overhead: FT-PDGESV (2 errors)
But ...

- Good
  - We have some algorithms that have invariants that make them resilient
- Bad!
  - An algorithm is not an application!
- How do we scale up the algorithm properties to the scale of the application?

\[ T_{L}^{\text{final}} = \frac{1}{1 - \frac{D + R_{L} + \text{Recons}^{\text{ABFT}}}{\mu}} \times (\alpha \times T_{L} + C_{L}) \]

Time (with overheads) of Library phase is constant (in \( P_{G} \)):

\[ T_{G}^{\text{final}} = \begin{cases} 
\frac{1}{1 - \frac{D + R_{G} + \text{Stays}}{\mu}} \times (T_{G} + C_{L}) & \text{if } T_{G} < P_{G} \\
(1 - \frac{C}{P_{G}})(1 - \frac{D + R + \frac{P_{G}}{\mu}}{\mu}) & \text{if } T_{G} \geq P_{G}
\end{cases} \]

Time (with overheads) of General phase accepts two cases:

Which is minimal in the second case, if

\[ P_{G} = \sqrt{2C(\mu - D - R)} \]
Mixed resilient solutions (model)

• An iterative application using a resilient library
  • Protect the application with traditional checkpoint/restart
  • Protect the library with new techniques (ABFT)

• Augment the initial data with extra information (e.g. checksum)
  • Maintain this extra information through the algorithm
  • Allow soft and hard error survival

• Library using ABFT: dense and sparse LA, matrix-matrix multiplications, one-sided and two-sided factorizations, CG, GMRES

≠ Exascale machine: same comp increase
Memory per component remains constant
Problem size increases \(O(\sqrt{n})\): matrix based
\(\mu\) at \(n=10^5\) 1 day is \(O(1/n)\)
\(C (=R)\) at \(n=10^5\) is 1m, is in \(O(n)\)
80% in library, 20% in application
Mixed resilient solutions (model)

- An iterative application using a resilient library
  - Protect the application with traditional checkpoint/restart
  - Protect the library with new techniques (ABFT)

- Augment the initial data with extra information (e.g. checksum)
  - Maintain this extra information through the algorithm
  - Allow soft and hard error survival

- Library using ABFT: dense and sparse
  LA, matrix-matrix multiplications, one-sided and two-sided factorizations, CG, GMRES

≠ Exascale machine: same comp increase
Memory per component remains constant
Problem size increases (O(√n): matrix based)
µ at n=10^5 1 day is O(1/n)
C (=R) at n=10^5 is 1m, is in O(n)
80% in library, 20% in application
Mixed resilient solutions (model)

- An iterative application using a resilient library
  - Protect the application with traditional checkpoint/restart
  - Protect the library with new techniques (ABFT)

- Augment the initial data with extra information (e.g. checksum)
  - Maintain this extra information through the algorithm
  - Allow soft and hard error survival
- Library using ABFT: dense and sparse
  LA, matrix-matrix multiplications, one-sided and two-sided factorizations, CG, GMRES

≠ Exascale machine: same comp increase
Memory per component remains constant
Problem size increases \( O(\sqrt{n}) \): matrix based)
\( \mu \) at \( n=10^5 \) 1 day is \( O(1/n) \)
\( C (=R) \) at \( n=10^5 \) is 1m, independent of \( n \) \( (O(1)) \)
80% in library, 20% in application
Mixed resilient solutions (model)

- An iterative application using a resilient library
  - Protect the application with traditional checkpoint/restart
  - Protect the library with new techniques (ABFT)

- Augment the initial data with extra information (e.g. checksum)
  - Maintain this extra information through the algorithm
  - Allow soft and hard error survival

- Library using ABFT: dense and sparse
  LA, matrix-matrix multiplications, one-sided and two-sided factorizations, CG, GMRES

≠ Exascale machine: same comp increase
Memory per component remains constant
Problem size increases \((O(\sqrt{n})\): matrix based)
\(\mu\) at \(n=10^5\) 1 day is \(O(1/n)\)
\(C (=R)\) at \(n=10^5\) is 1m, independent of \(n\) \((O(1))\)
80% in library, 20% in application
Fenix: Exploring Automatic, Online Failure Recovery for Scientific Applications at Extreme Scales

Marc Gamell\textsuperscript{1}, Daniel S. Katz\textsuperscript{2}, Hemanth Kolla\textsuperscript{3}, Jacqueline Chen\textsuperscript{3}, Scott Klasky\textsuperscript{4}, Manish Parashar\textsuperscript{1}

\textsuperscript{1}Rutgers Discovery Informatics Institute (RDI\textsuperscript{2}), Rutgers University
\textsuperscript{2}University of Chicago & Argonne National Laboratory
\textsuperscript{3}Sandia National Laboratories
\textsuperscript{4}Oak Ridge National Laboratory

Key contributions

- On-line, local, semi-transparent recovery from process, node, blade and cabinet failures
- Targets MPI-based parallel applications
- Using application-specific double in-memory, implicitly coordinated, local checkpoints

Approach

Fenix
- Design and implementation of the approach
- Deployed on Titan Cray XK7 at ORNL

Implementation details

- Built on top of MPI-ULFM
- Tested up to - 8192 cores w/ failures
- 250k cores w/o failures
- Provides C, C++ and Fortran interfaces

Experimental Evaluation

- S3D combustion numerical simulation
- Sustained performance with MTBF = 47 seconds
- Experiments inject real failures
4. Recovering from high-frequency failures

Conclusions:
• Online recovery allows the usage of in-memory checkpointing, $O(0.1s)$
• Efficient recovery from high frequency node failures, as exascale compels
• With failures injected every 189, 94 and 47 seconds, the total job run-time penalty is 10%, 15% and 31%, respectively
  • Note that current production runs’ fault tolerance cost is 31%!
• This can dramatically improve by optimizing ULFM shrink
Current C/R production code exhibits a 31% overhead with 6 failures a day.
Resilient X10

• X10 is a PGAS programming language
  • Need for resilience forced the use of TCP

Happens Before Invariance Principle (HBI): Failure of a place should not alter the happen 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*/}
```

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 receives according to probes
  • Asynchronous background collective operations (on multiple different communicators to form 2D grids, etc).

• Recovery
  • Upon failure all communicators recreated (shrink + spares, or using MPI_COMM_SPAWN to create new processes)
  • Ranks reassigned identically to rebuild the same X10 “teams”

• FT layer
  • Unnecessary, X10 has a runtime that hides all MPI from the application and handles failures internally

Source: Sara Hamouda, Benjamin Herta, Josh Milthorpe, David Grove, Olivier Tardieu. Resilient X10 over Fault Tolerant MPI. In: poster session SC’16, Austin, TX
Other users activities around ULFM


- **ENGELMANN, Christian et NAUGHTON, Thomas.** A NETWORK CONTENTION MODEL FOR THE EXTREME-SCALE SIMULATOR.


Frameworks using ULFM : LFLR, FENIX, FTLA, FALANX

Programming models: X10, CoArray
Conclusion

- ULFM is not a fault management approach
  - It’s a toolbox to build higher-level application/domain specific techniques
    - detection / revoke / agreement
    - Support for numerous usage patterns (in-place C/R, iterative convergence, master/worker, ABFT, message logging...)

- There are now viable alternatives to handling the faults with C/R
  - HPC applications can definitively benefit
  - But more importantly this makes MPI a suitable programming environment outside HPC

- These concepts are not unique to MPI, they are a needed underlying foundation for any parallel programming paradigm
More info, resources

http://fault-tolerance.org/

• Standard draft document
  • https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/323

• Implementation(s) available
  • Version 1.1 based on Open MPI 1.6 released early 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
  • Runs with ALPS, PBS, etc...
  • RMA, I/O in progress
  • Version 2.0 based on Open MPI 2.0 to be released Q2 2016
    https://bitbucket.org/icldistcomp/ulfm
  • Full communicator-based (point-to-point and all flavors of collectives) support
  • Network support IB, uGNI, TCP, SM
  • Runs with PMiX and also with ALPS, PBS, etc...
  • RMA, I/O