FT Working Group Ticket #276: Run-Through Stabilization Process Fault Tolerance

Joshua Hursey

Postdoctoral Research Associate
Oak Ridge National Laboratory
hurseyjj@ornl.gov
http://users.nccs.gov/~jjhursey

MPI Forum – July 18, 2011

Fault Tolerance Working Group

Define a set of semantics and interfaces to enable fault tolerant applications and libraries to be portably constructed on top of MPI.

- **Application/Library involved fault tolerance** (not transparent)
- ***fail-stop* process failures:**
  - A process failure in which the MPI process is permanently stopped, often due to a component crash.
- **Two Complementary Proposals:**
  - **Run-Through Stabilization:** Ticket #276 – Target MPI 3.0
    - Continue running and using MPI even if one or more MPI processes fail
  - **Process Recovery:** Ticket TBD – Target MPI 3.1
    - Replace MPI processes in existing communicators, windows, file handles

Run-Through Stabilization Proposal

- **Error Handlers:**
  - **Application/Library <u>must *opt-in*</u> by:**
    - Replacing `MPI_ERRORS_ARE_FATAL` with at least `MPI_ERRORS_RETURN`
  - **MPI implementation <u>may *opt-out*</u> by:**
    - Returning `MPI_ERR_UNSUPPORTED_OPERATION` for new operations, and
    - Never returning the new error class `MPI_ERR_RANK_FAIL_STOP`
- **Error Class:** `MPI_ERR_RANK_FAIL_STOP`
  - **A process in the operation is failed** (fail-stop failure)
  - **If this error class is returned then the MPI agrees to provide the specified semantics and interfaces defined by this proposal**
- **The behavior of MPI after returning other error classes remains undefined by the standard.**

Run-Through Stabilization Proposal

- **Failure detector exposed to the application:**
  - ***Perfect Detector* = *strongly accurate & complete***
    - No process is reported as failed before it actually fails
    - Eventually every failed process will be known to all processes
- **Process failures are managed on a per-{group, communicator, window, file handle} basis**
  - All such objects remain active across failures
  - Object preservation is important to library development

40 New MPI Operations

- **Validation:** (34) Update, access, and modify process state

```
/**** Local List Scope ****/
MPI_{Group,Comm,Win,File}_validate                 - Local
MPI_{Group,Comm,Win,File}_validate_get_num_state   - Local
MPI_{Group,Comm,Win,File}_validate_get_state       - Local
MPI_{Group,Comm,Win,File}_validate_get_state_rank  - Local
MPI_{Comm,Win,File}_validate_set_state_null        - Local
/**** Global List Scope ****/
MPI_{Comm,Win,File}_validate_all                   - Collective
MPI_{Comm,Win,File}_ivalidate_all                  - Collective (Non-Blocking)
MPI_{Comm,Win,File}_validate_all_get_num_state     - Local
MPI_{Comm,Win,File}_validate_all_get_state         - Local
MPI_{Comm,Win,File}_validate_all_get_state_rank    - Local
```

- **Other:** (6)

```
/**** Error Handler Comparison ****/
MPI_Errhandler_compare                     - Local
/**** Remote Termination ****/
MPI_Comm_kill                              - 1 sided
/**** Collectively Active ****/
MPI_{Comm,File}_is_collectively_active     - Local
/**** MPI_Rank_info Language Binding ****/
MPI_Rank_info_{f2c,c2f}                    - Local
```

MPI_Rank_info Type

- `MPI_Rank_info` **is a semi-opaque type (like** `MPI_Status`**)**
  - `info.MPI_RANK` : **Rank in the specified process group**
  - `info.MPI_STATE` : **State of the rank in the process group**
  - `info.MPI_FLAGS` : **Implementation specific modifiers**
- **Process State can be one of the following:**
  - `MPI_RANK_STATE_OK` : **Normal, running state**
  - `MPI_RANK_STATE_FAILED` : **Unrecognized fail-stop failure**
  - `MPI_RANK_STATE_NULL` : **Recognized fail-stop failure**
- **Application** ***recognized*** **fail-stop process failures provide** `MPI_PROC_NULL`**-like semantics.**

Quick Overview of Semantics

- **Communication Object Creation:**
  - **Uniformly created across collective group**
- **Point-to-Point**
  - **Isolation of failures:**
    Proc. A can communicate with Proc. B, even if Proc. C has failed
- **Collectives**
  - **Must be at least *Fault-Aware*:**
    Cannot 'hang' in the presence of process failure, but do not need to return the same return code everywhere
  - **May be *Fault-Tolerant*:**
    Fault-Aware and provides uniform return codes at all processes

Performance Notes: Open MPI Prototype

- **NetPIPE: Shared Memory**
  - Latency: 0.84 to 0.85 microseconds (1.2%)
  - Bandwidth: 8957 to 8920 Mbps (0.4%)

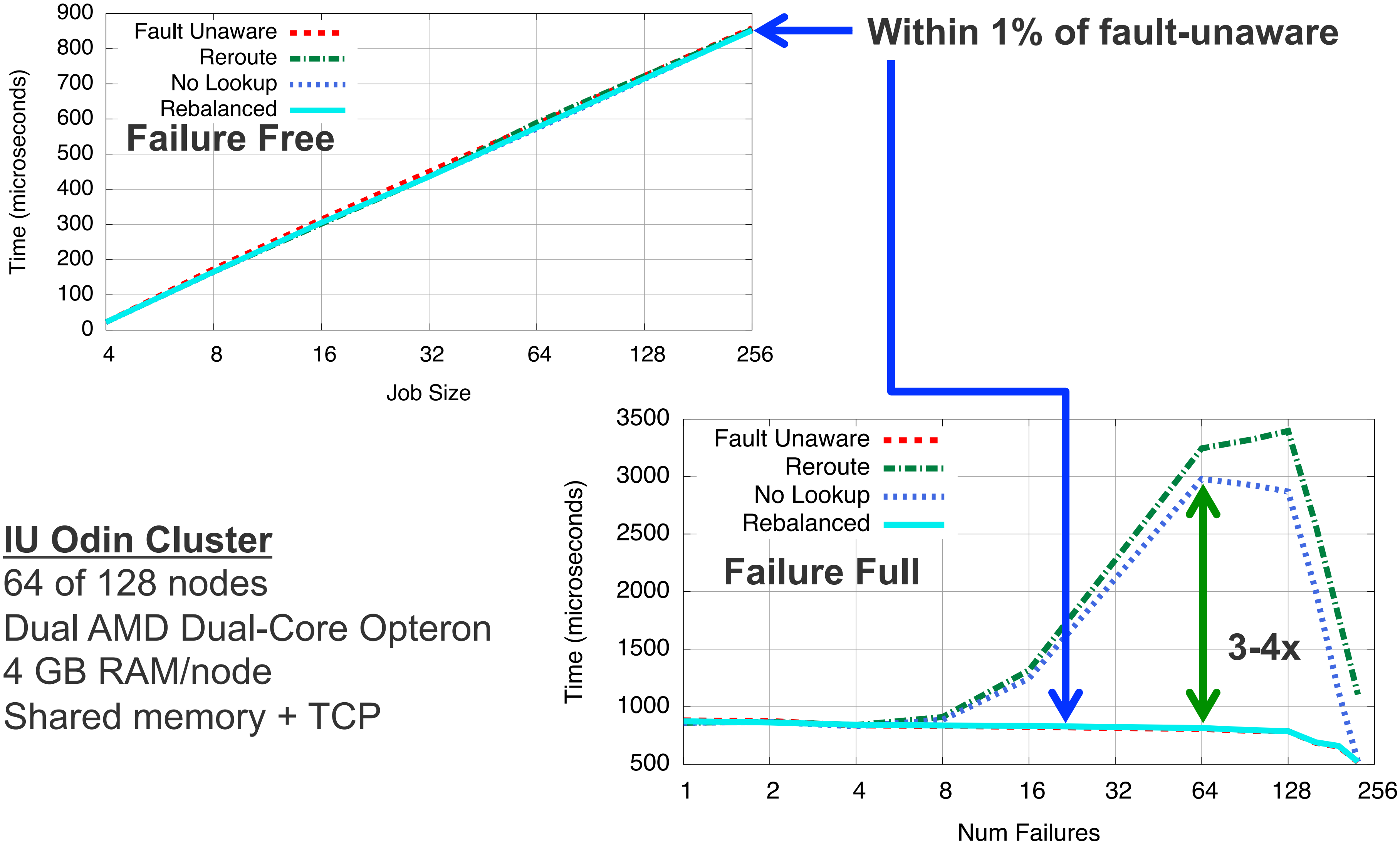
- **Collectives: Fault-Aware**
  - MPI_Barrier:
    Within 1% of fault-unaware, regardless of # failures

    Hursey, J., Graham, R., "*Preserving Collective Performance Across Process Failure for a Fault Tolerant MPI*," HIPS Workshop @ IPDPS, 2011.

  - MPI_Comm_validate_all:
    Within 3% of MPI_Allreduce() collective, log-scaling

    Hursey, J., Naughton, T., Valle, G., Graham, R., "*A Log-Scaling Fault Tolerant Agreement Algorithm for a Fault Tolerant MPI*," EuroMPI, 2011 (to appear).

MPI_Barrier: fault-aware collective, binomial tree



IU Odin Cluster

64 of 128 nodes

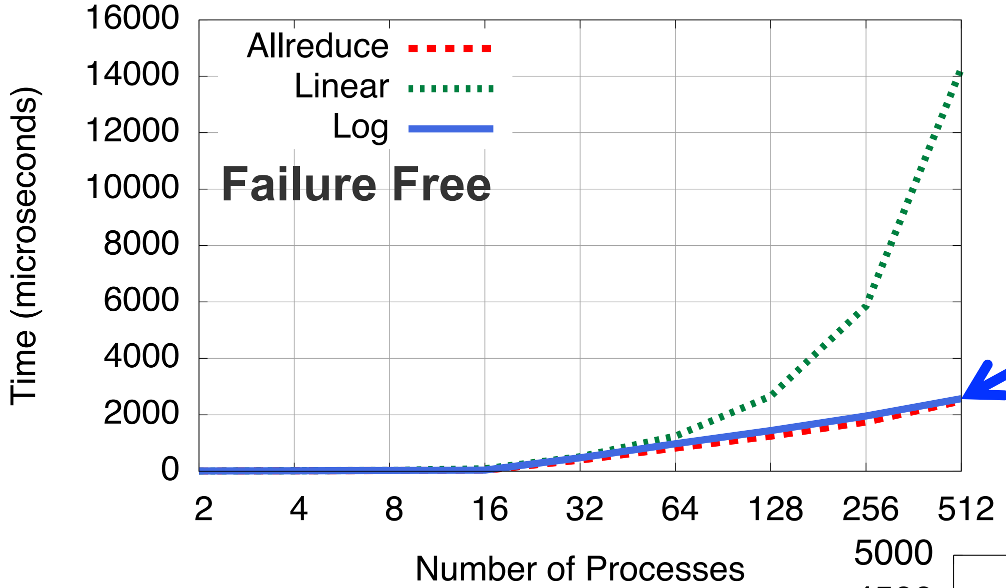
Dual AMD Dual-Core Opteron

4 GB RAM/node

Shared memory + TCP

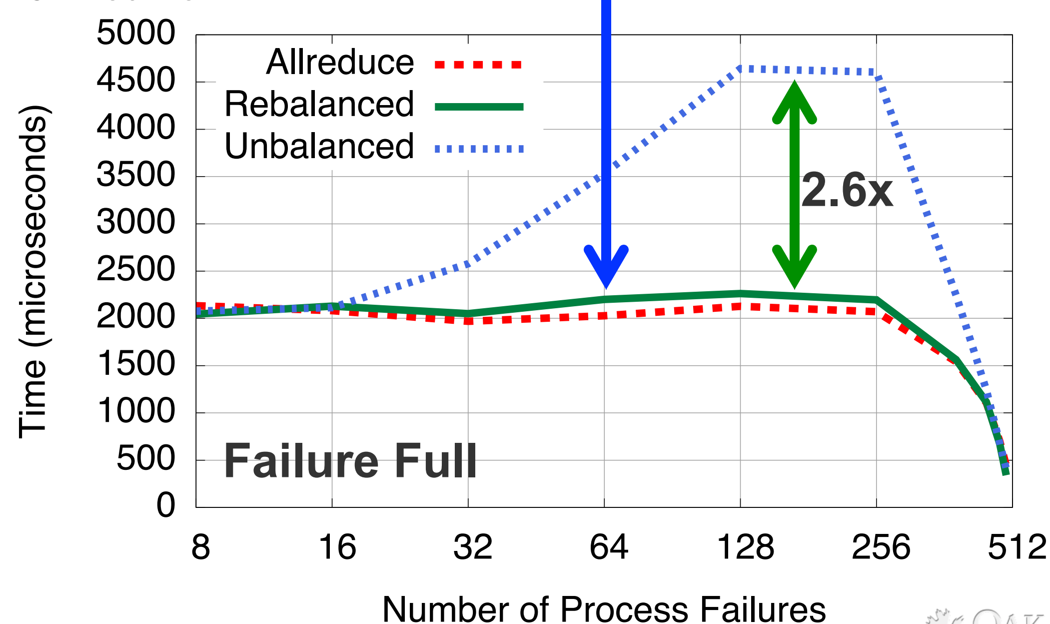
Hursey, J., Graham, R., “*Preserving Collective Performance Across Process Failure for a Fault Tolerant MPI*,” HIPS Workshop @ IPDPS, 2011.

MPI_Comm_validate_all: 2-phase commit protocol, binomial tree



At worst 3% of Allreduce

At worst 6% of Allreduce



<u>ORNL Smoky Cluster</u>

32 of 80 nodes

Four AMD Quad-Core Opteron

32 GB RAM/node

Shared memory + TCP

Hursey, J., Naughton, T., Valle, G., Graham, R., "*A Log-Scaling Fault Tolerant Agreement Algorithm for a Fault Tolerant MPI*," EuroMPI, 2011 (to appear).

Application Example: (NOAA)

Weather Forecasting

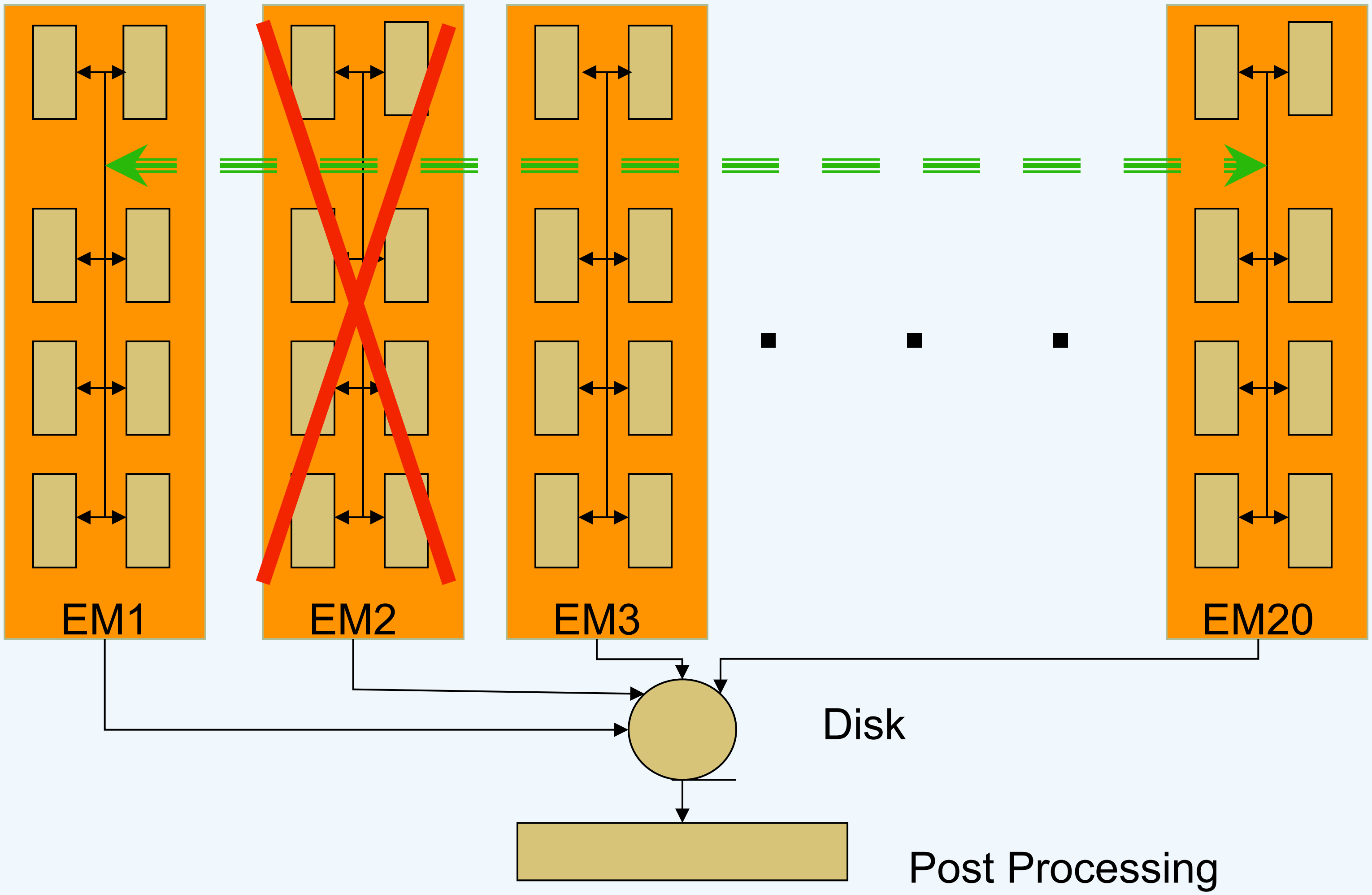
NOAA's Primary Use Case

- Operational case
  - Fault-tolerance, not fault-recovery - upon a failure, permit surviving ensemble members to complete
    - Deadline processing not real-time processing
    - Must be able to set max timeout for ensemble member failure detection/declaration
    - Prune and continue

How It Might Work

- Until recently, each ensemble member was a separate "binary", separate but simultaneous launch. With MPI3 a single binary with separate communicators for each ensemble member would be used.
- Only one MPI task per ensemble member would perform the fault-tolerant rendezvous process.
- Any ensemble member failure takes out the entire ensemble member.
- Only one ensemble member at a time is the "master" to focus the rendezvous (first ensemble member first, simple/direct succession in case of failure).
- A parameter value will set the maximum time to wait for an ensemble member to rendezvous.
- All communication from "clients" to "master" will include an acknowledge (ack) back to the client containing information regarding the result of the rendezvous.
- "Master" and "Client" are really identical, just minor execution differences. All ensemble members capable of becoming "Master".

Future - Failure Result

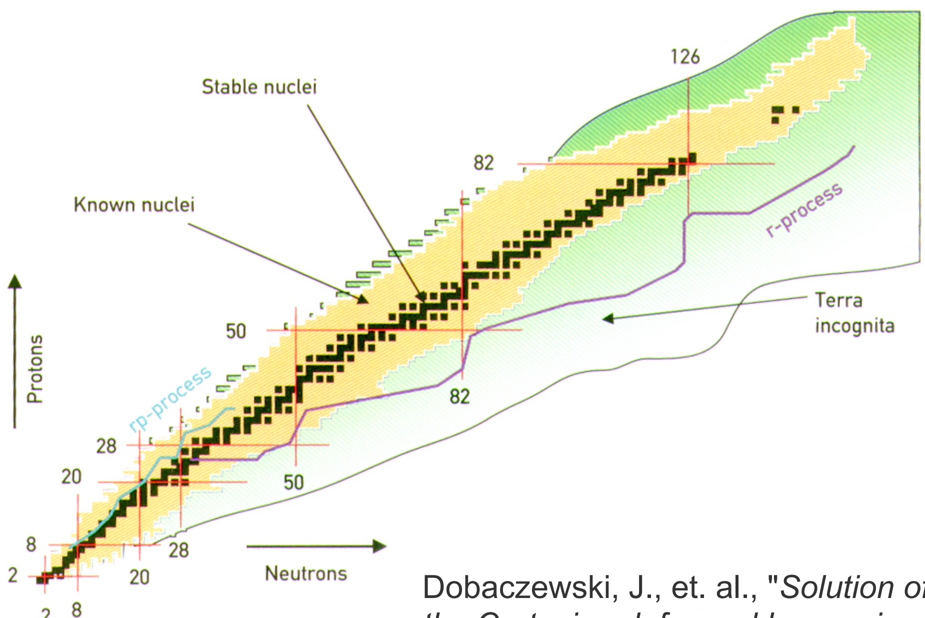


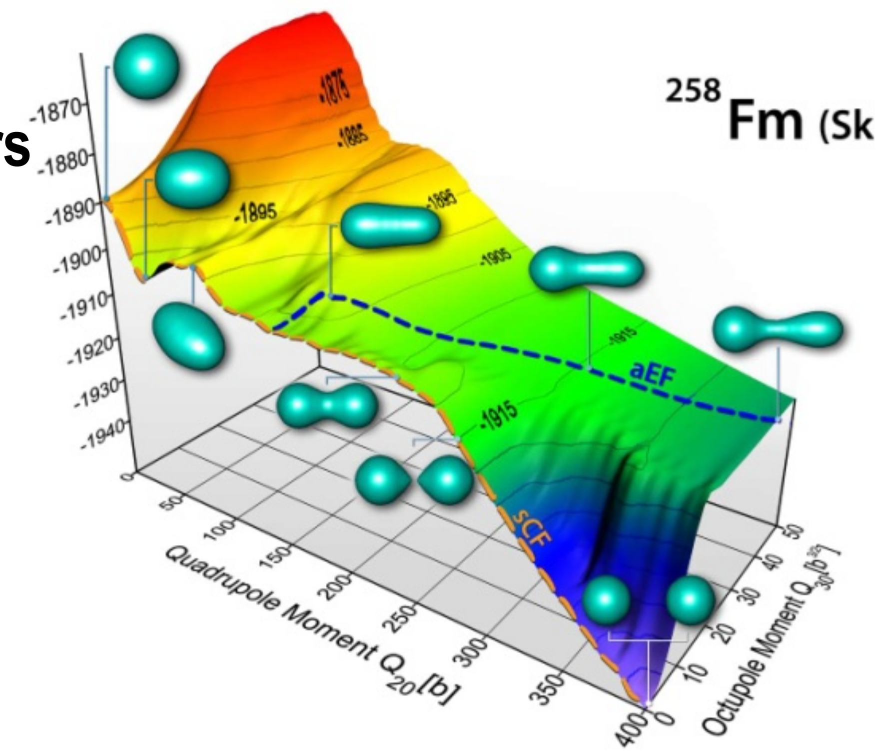
Application Example: (LLNL/ORNL)

HFODD – Nuclear Physics

Application Example: (LLNL/ORNL) HFODD

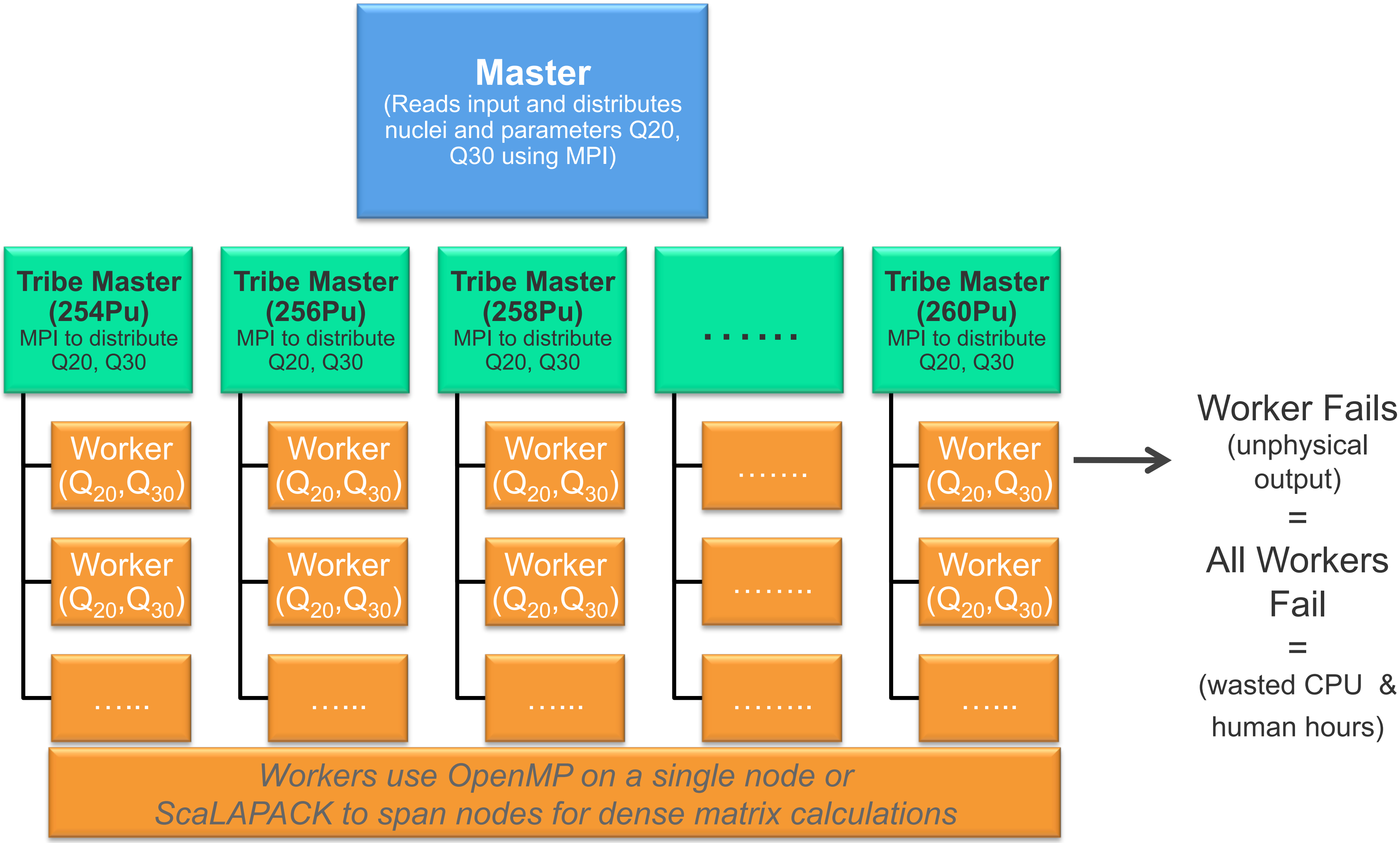
- **Solves the Hartree-Fock Bogoliubov equations in deformed, Cartesian harmonic oscillator coordinates**
- **Systematic calculations involve**
  - **Thousands of nuclei**
  - **Hundreds of possible parameters**





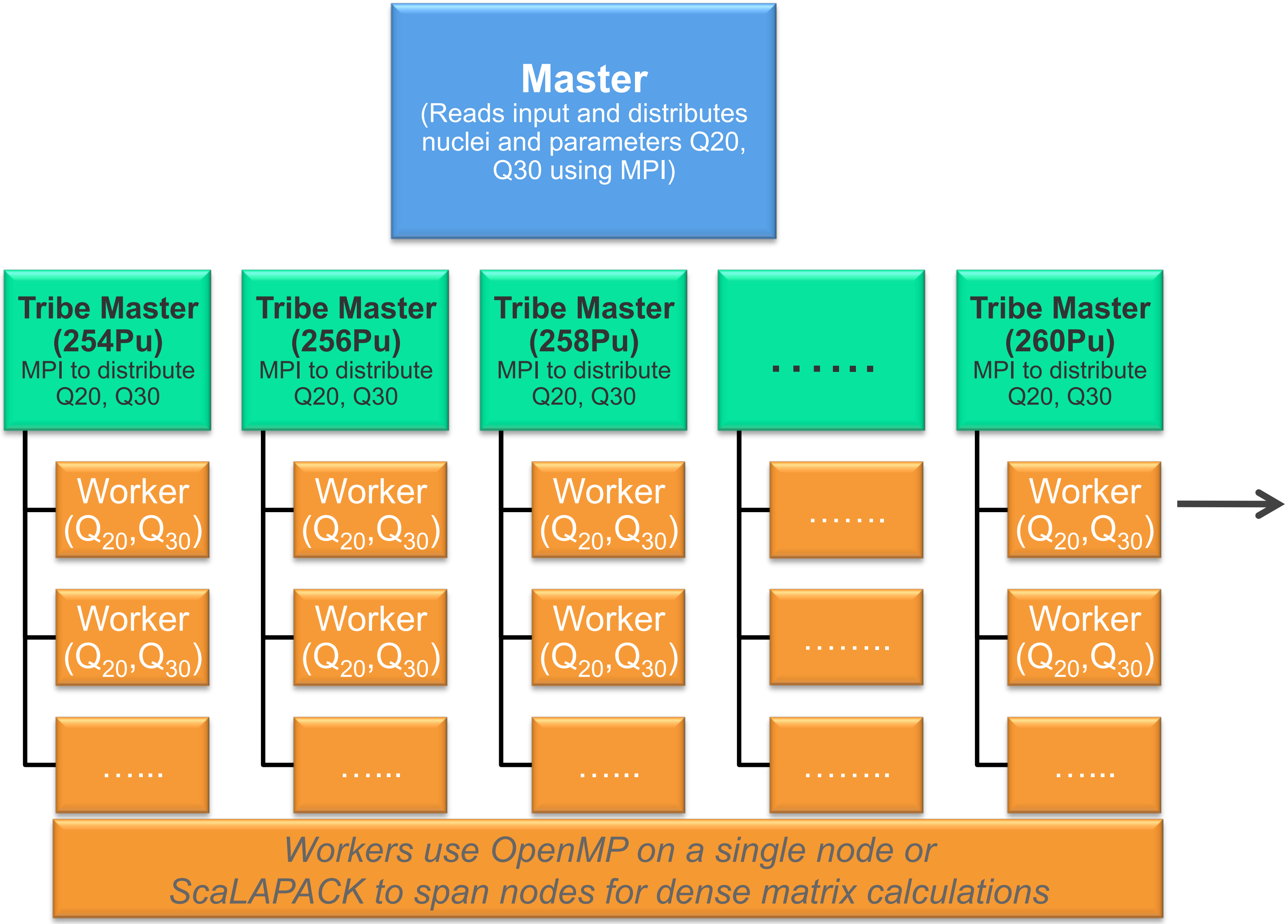
Dobaczewski, J., et. al., "*Solution of the Skyrme-Hartree-Fock-Bogolyubov equations in the Cartesian deformed harmonic-oscillator basis. (VI) HFODD (v2.40h): a new version of the program*", Computer Physics Communications, 2009.
http://www.fuw.edu.pl/~dobaczew/hfodd/hfodd.html

Application Example: (LLNL/ORNL) HFODD Run Characterization (Current)



http://www.fuw.edu.pl/~dobaczew/hfodd/hfodd.html

Application Example: (LLNL/ORNL) HFODD Run Characterization (Desired)



If worker fails

Then worker gets new (Q20, Q30) from Tribe Master

-- and/or ---

If critical # of workers fail

Then Tribe Master kills all workers and begin calculations on a new nuclei

http://www.fuw.edu.pl/~dobaczew/hfodd/hfodd.html

Application Example: (A Sampling of Others) General Algorithm Based Fault Tolerance

- Rob T. Aulwes, "**Integrating Fault Tolerance into the Monte Carlo Application Toolkit**," Resilience Summit @ LACSS, 2010.
- T. Davies, C. Karlsson, H. Liu, C. Ding and Z. Chen, "**High Performance Linpack Benchmark: A Fault Tolerant Implementation without Checkpointing**," International Conference on Supercomputing, 2011.
- D. Hakkarinen and Z. Chen, "**Algorithmic Cholesky factorization fault recovery**," In Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium, 2010.
- Z. Chen and J. Dongarra, "**Algorithm-based fault tolerance for fail-stop failures**," IEEE Transactions on Parallel and Distributed Systems, 2008.
- H. Ltaief, E. Gabriel, and M. Garbey, "**Fault tolerant algorithms for heat transfer problems**," Journal of Parallel and Distributed Computing, 2008.
- Y. Du, P. Wang, H. Fu, J. Jia, H. Zhou, and X. Yang, "**Building single fault survivable parallel algorithms for matrix operations using redundant parallel computation**," International Conference on Computer and Information Technology, 2007.
- J. Langou, Z. Chen, G. Bosilca, and J. Dongarra, "**Recovery patterns for iterative methods in a parallel unstable environment**," SIAM Journal of Scientific Computing, 2007.
- C. Engelmann and A. Geist, "**Super-scalable algorithms for computing on 100,000 processors**," in Proceedings of International Conference on Computational Science, 2005.
- K.-H. Huang and J. A. Abraham, "**Algorithm-based fault tolerance for matrix operations**," IEEE Transactions on Computers, 1984.
- B. Randell, "**System structure for software fault tolerance**," in Proceedings of the international conference on reliable software, 1975.