

Using ULFM to design resilient numerical schemes on HPC platforms

By AMIROUCHE Said

Supervised by : GIRAUD Luc & AGULLO Emmanuel





Outline

- Introduction
- Hard fault recovery in distributed memory environment
- The ULFM proposition
- Experimental example
- Conclusion
- Prospects



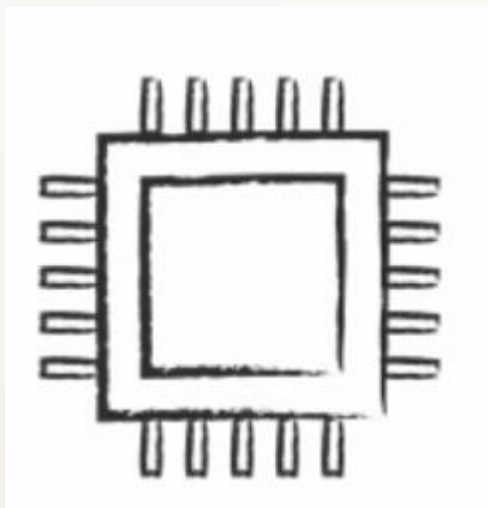
Internship context

- 6 months internship in HiePACS team – INRIA (Bordeaux).
- Researchs on parallel algorithms for challenging numerical simulations
- Luc Giraud & Emmanuel Agullo From INRIA as supervisors.
- Project collaboration with ULFM team from University of Tennessee presented by George Bosilca and University of Manchester by Mawussi Zounon.



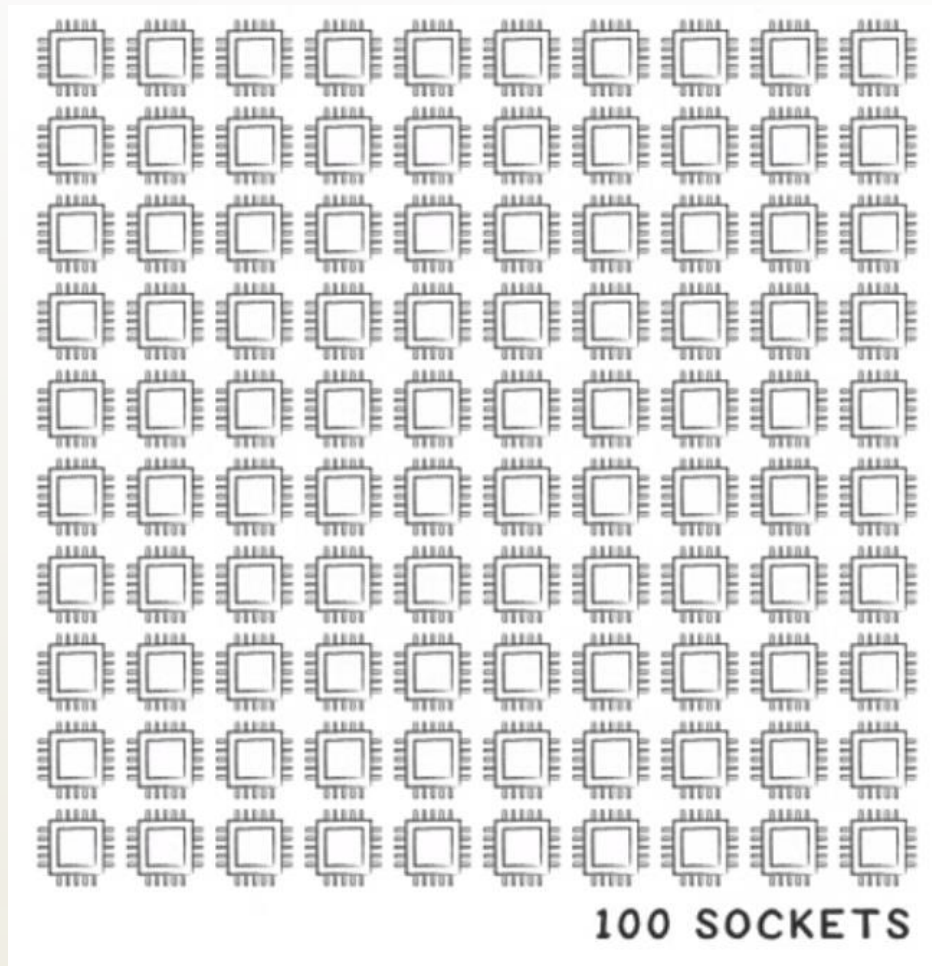
Introduction

- Research on HPC resilience for large scale numerical simulations
- Failures are due to memory, software and hardware errors
- Failures rate increase proportionnaly with the plateforme scale



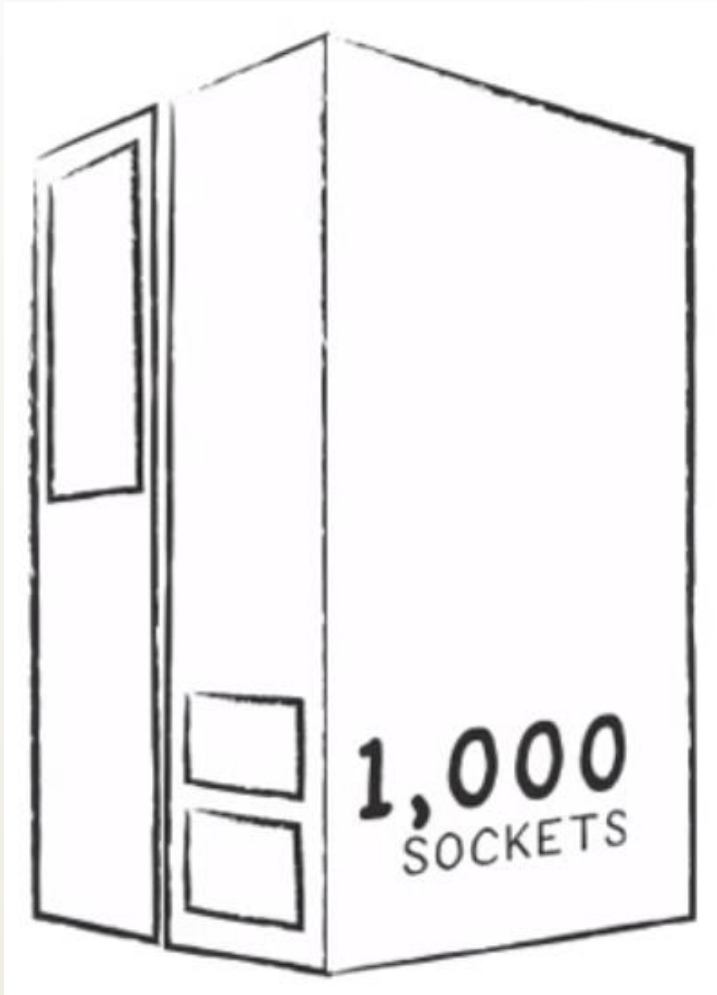
100
YEARS

MEAN TIME
BETWEEN FAILURES



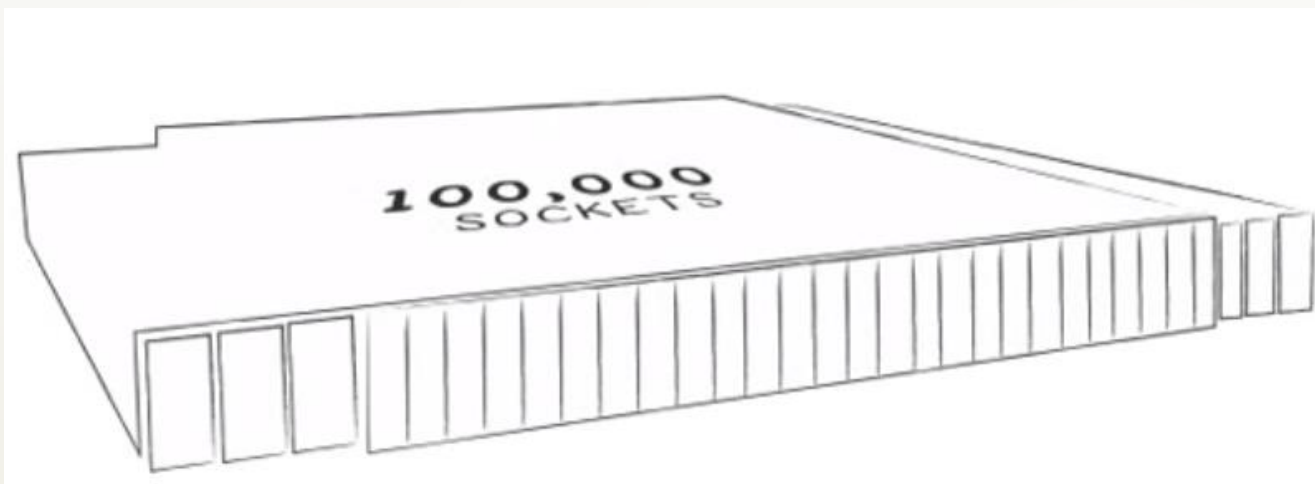
1
YEAR

MEAN TIME
BETWEEN FAILURES



36
DAYS

MEAN TIME
BETWEEN FAILURES



8
HOURS

MEAN TIME
BETWEEN FAILURES



Introduction

- Research on HPC resilience for large scale numerical simulations
- Failures are due to memory, software and hardware errors
- Failures rate increase proportionnaly with the platforme scale
- MPI provides pratically no support for fault tolerance
- ULFM Proposed changes to MPI specification to define necessary set of enhancements for supporting fault-tolerance



Hard fault recovery in distributed memory environnement

Checkpoint / Restart

- Store the whole state of computation on disk space or remote memory
- Roll back to a previous saved state and continue working
- Advantage
 - Resume work from an advanced state without losing results
- Defaults
 - Additional time/ressources for saving and restarting
 - An application will no longer progress If $MTBF < \text{application restart time}$





Replication

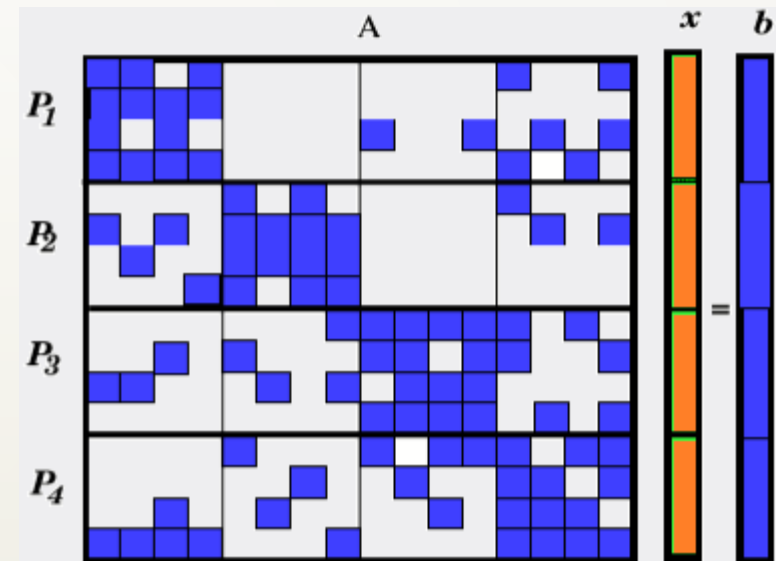
- Use « wasted » ressources in the cluster machines to run multiple copies of the same application simultaneously.
- In case of failure, a vote system is made between the different copies by selecting the most common results
- Advantage
 - Assure the accuracy of the results even with failures
- Defaults
 - Not applicable when the system utilisation is larger then 50 %

Algorithm-Based Fault Tolerance (ABFT)

- The numerical algorithms are modified to include methods for detecting and correcting errors
- Develop new algorithms that are naturally resilient to faults
- Advantage
 - The time to solution is often slightly larger in the presence of faults
- Default
 - An impact in terms of some loss of accuracy
 - Revisited each algorithm individually is required

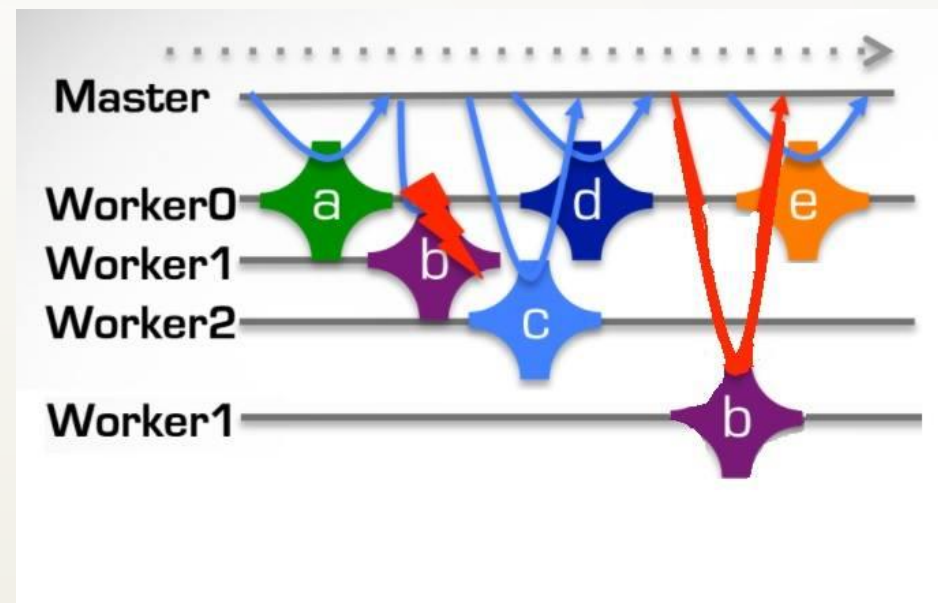
Numerical recovery strategies

- Iteratively solve sparse linear systems
- We distinguish two types of data
 - **Static data**
 - **Dynamic data**
- Generic iterative loop: neighbor-neighbor communication, local computation, global reduction for convergence check

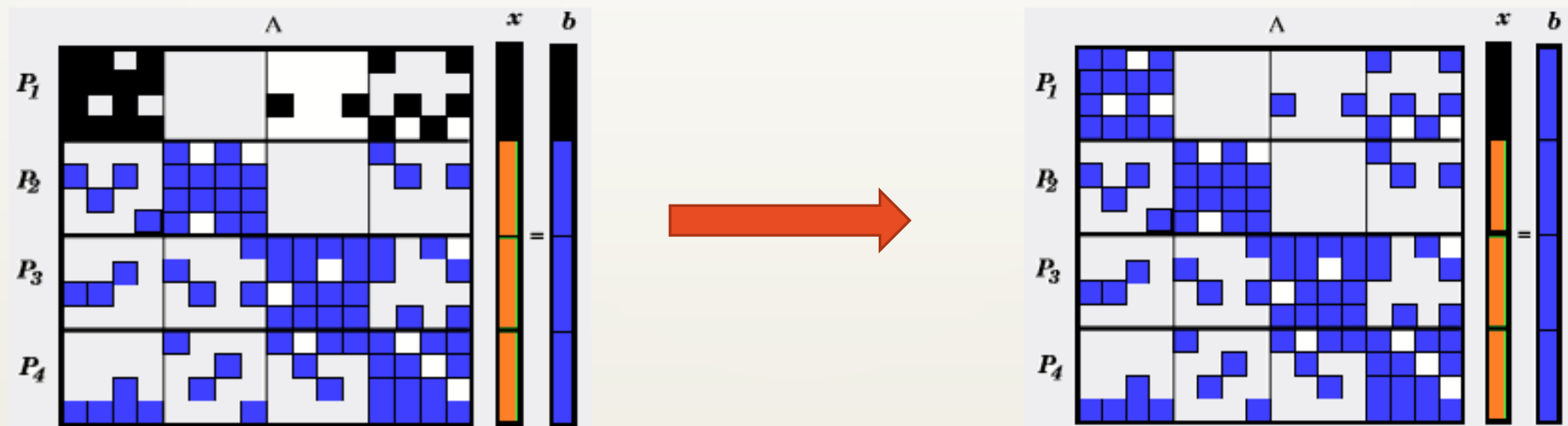


RESET

- Determine the dead process's rank
- Re-initialize its static data
- Continue working

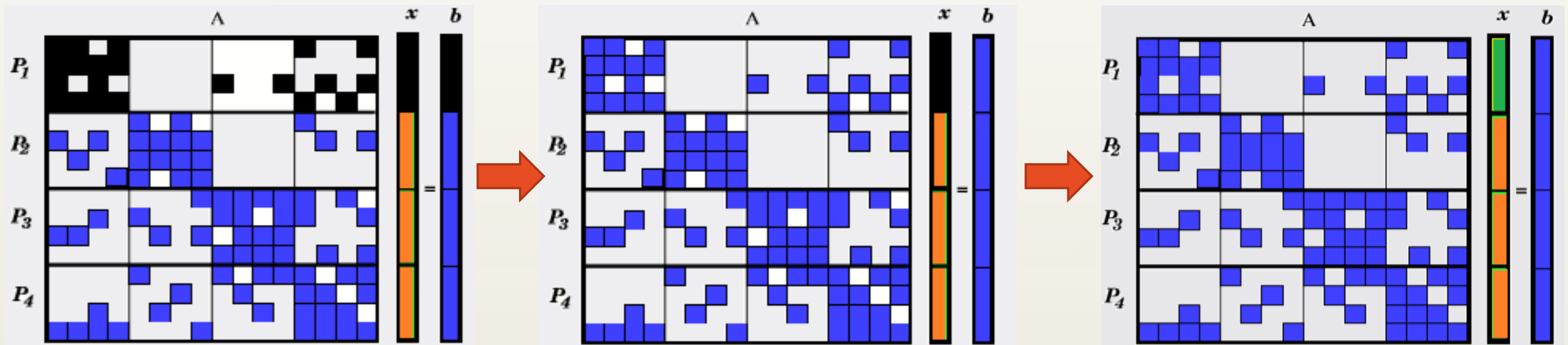


- Set A and b to their initial values for the case of $Ax = b$



Linear interpolation

- Determine the dead process's rank
- Reset its static data
- Interpolate its dynamic data by computing it locally using the dynamic data of the processes in communication with it



Linear interpolation (1)

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} ? \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \text{ How to interpolate } x_1?$$

$$\text{Solve } A_{11}x_1 = b_1 - A_{12}x_2$$

The ULFM Proposition

- Acronyme for **U**ser **L**evel **F**ailure **M**itigation
- Developed by MPI Fault Tolerance Working Group
- ABFT can easily be implemented on the top of ULFM
- Semantics include
 - the detection and identification of the failure process
 - Propagating the failure information on the survival processes

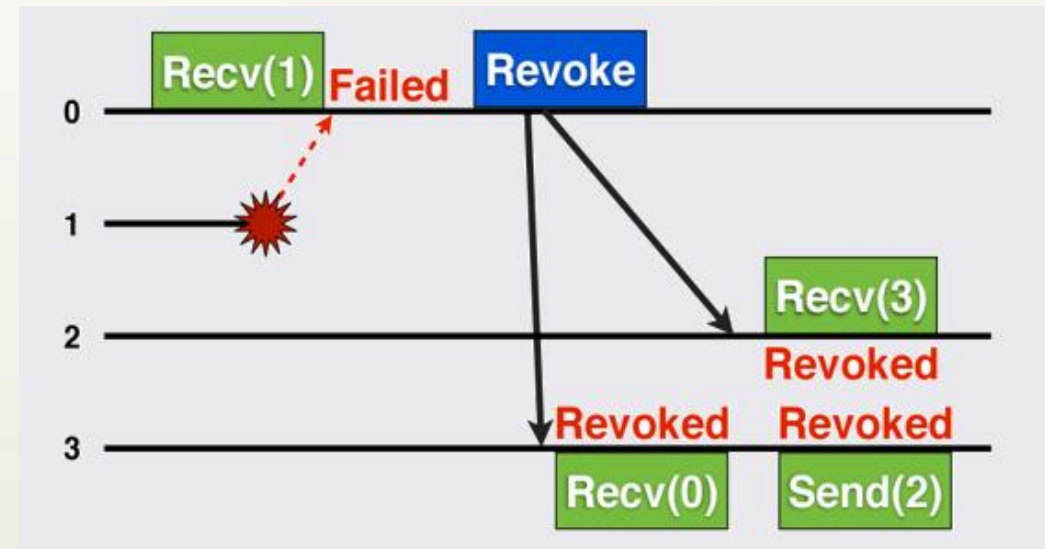


The ULFM Proposition

- Operations that can't complete return `ERR_PROC_FAILED`
- Operations that can be completed return `MPI_SUCCESS`
- New constructs `Comm_Revoke` resolves inconsistencies introduced by failures
- Use existing MPI error handlers to take advantage of ULFM
 - By default, set to `MPI_ERRORS_FATAL`
 - Must change communicator's handler to `MPI_ERRORS_RETURN`
 - Make the necessary modifications in the code source for failure detection and fixing

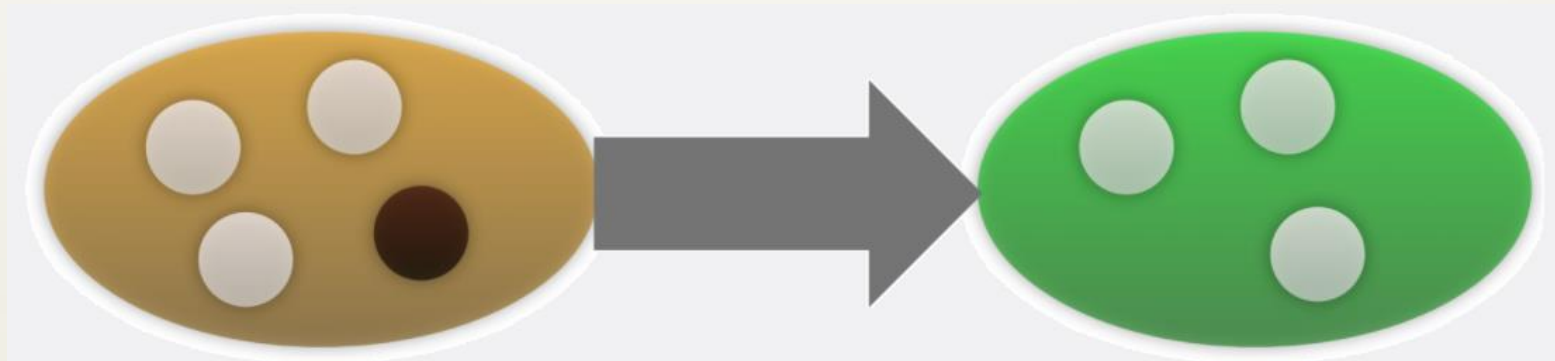
ULFM API : Communicator Revocation

- Method for revoking a communicator & propagate failure notices
- Must call `MPI_Comm_shrink()` to obtain usable communicator



ULFM API : Communicator Shrink

- Collective operation that removes any failed ranks during call and returns consistent “new” communicator at all sites
- Analogous to `MPI_Comm_split()` with “live” ranks in `newcomm`





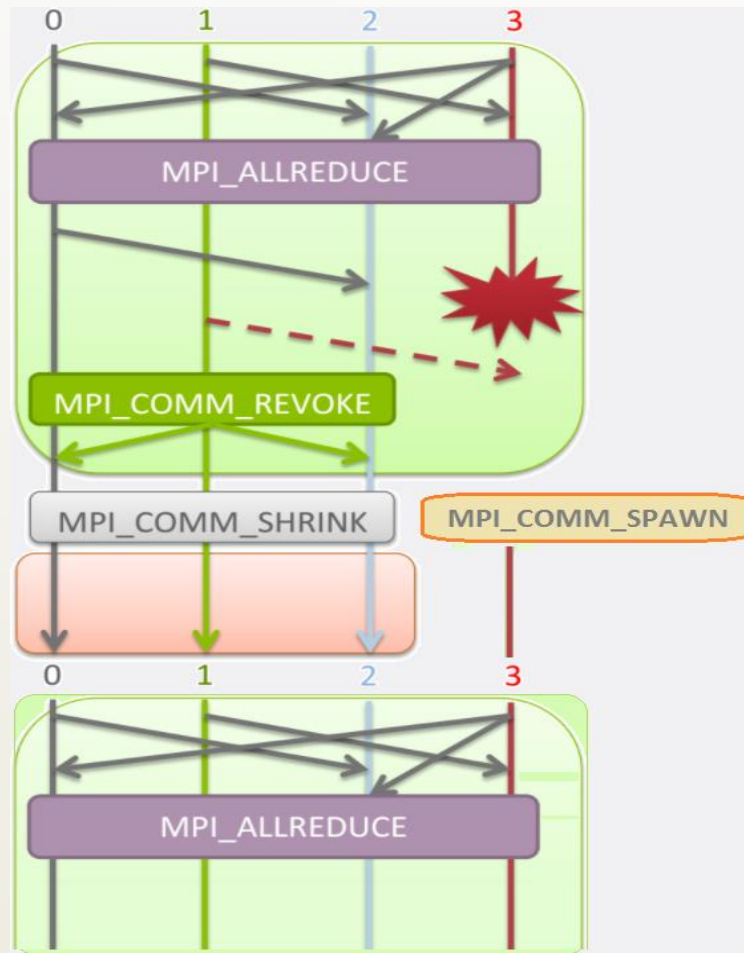
ULFM API : Agreement

- Performs fault tolerant agreement algorithm over a boolean flag
- Ignores all the failed processes
- Propagates the `MPI_ERR_Revoked` error code if it was called before or during the agreement
- Expensive and should be used sparingly

ULFM API: Quick Summary

- Use MPI error handlers
 - MPI_ERRORS_RETURN
- MPI Error classes
 - MPI_ERR_PROC_FAILED
 - MPI_ERR_REVOKED
 - MPI_ERR_PENDING
- New MPI functions
 - MPI_COMM_FAILURE_ACK
 - MPI_COMM_FAILURE_GET_ACKED
 - MPI_COMM_SHRINK
 - MPI_COMM_REVOKE
 - MPI_COMM_AGREE

An example of ULFM with ABFT



- Iterations with reduction
- After failure, revoke communicator and shrink it
- Spawn process to replace the failed ones
- Enter in numerical recovery phase for the dead processes

Experimental example

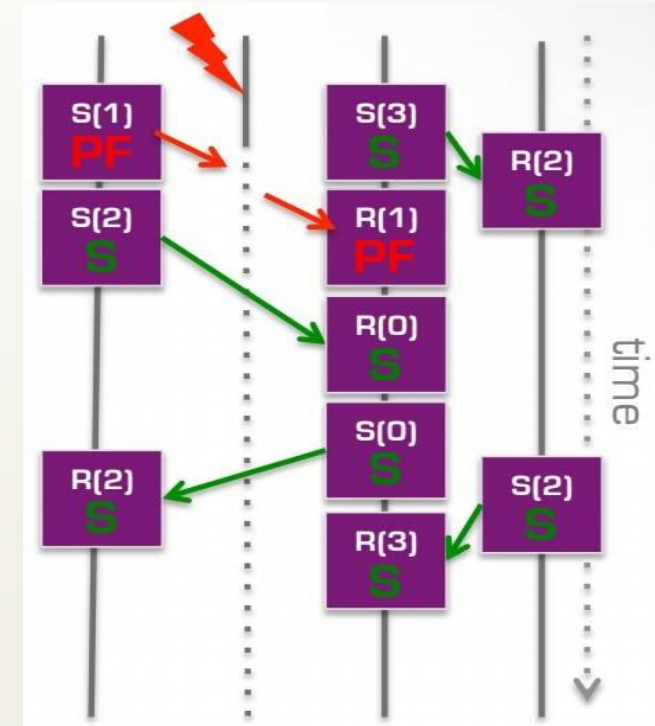
- Jacobi solution of a 2D PDE problem
 - Laplace equation in 2D with finite differences
 - Communication between neighbors for ghost-points exchange
- Update jacobi's problem to support ULFM
 - Change error handler to `MPI_ERRORS_RETURN`
 - Inject errors in different iterations, by killing a process each time
 - Check critical communications between processes, on error a failure notification is sent to all survival processes and all of them enter in communicator construction's step
 - Establish the value of the dead/respawned rank for its numerical recovery

Generate the faults

- Faults can originate from many sources : Memory, software, and hardware errors
- We cannot create such errors by ourselves
- Generate exterior and interior faults
 - Interior faults : simulation of process suicide.
 - Exterior faults : killing the process from the terminal.
- Simulate the different possible scenarios to see how it behaves

Failure detection & propagation

- Check Send/recv communications
- Check Allreduce
- Use MPI_COMM_AGREE to Propagate the MPI_ERR_REVOKED error code to all survival processes
- Default
 - costly method by checking critical communications in each step



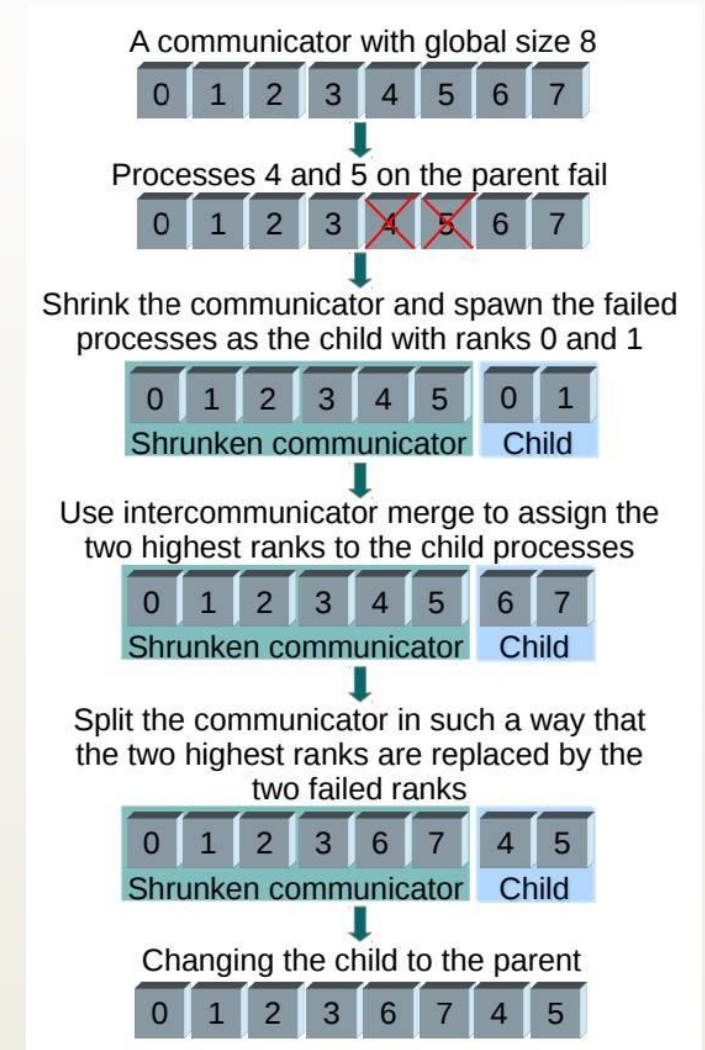


Failure recovery

- Communicator construction
- Numerical recovery strategies

Communicator construction

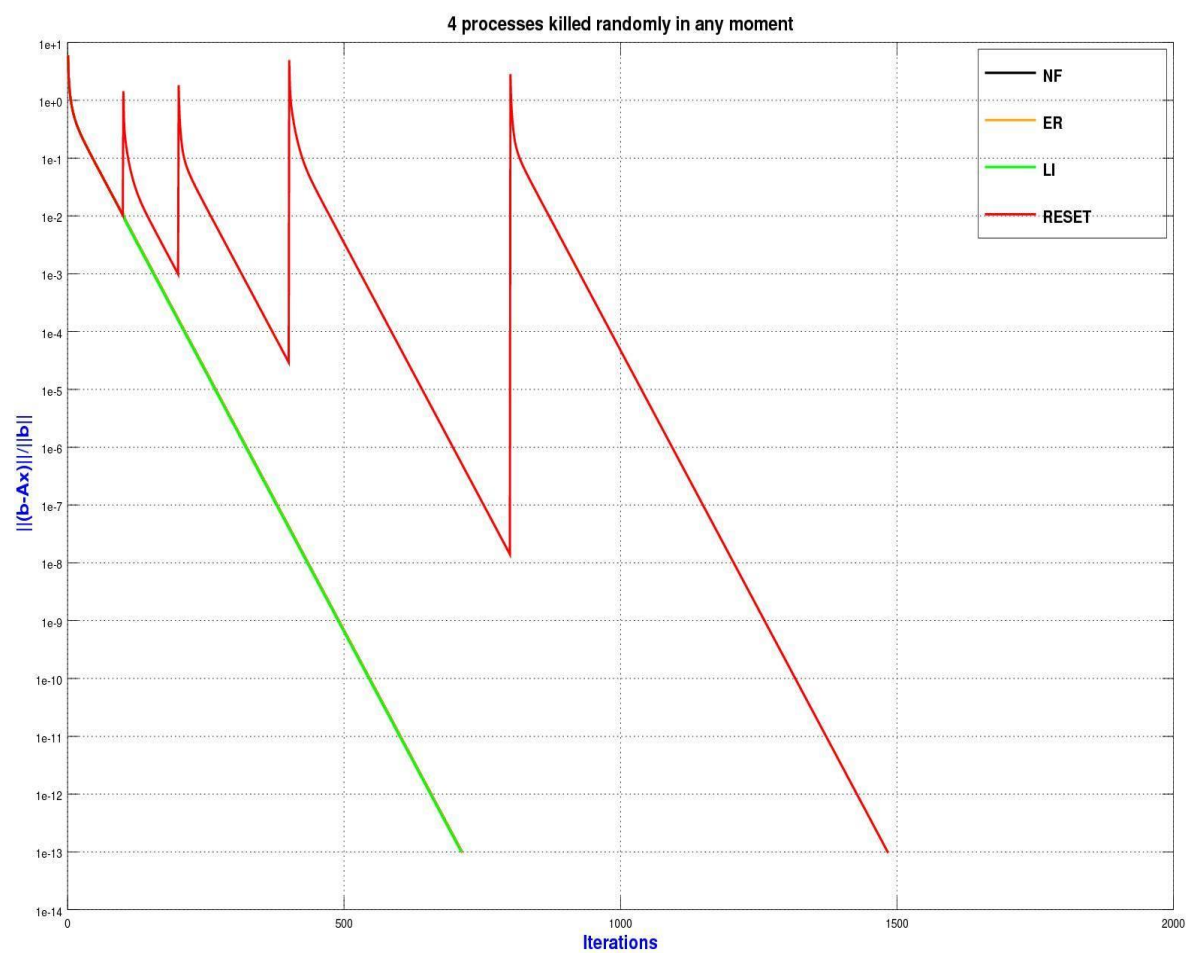
- Get knowledge of the failure processes and their ranks
- Shrink the communicator to let only the survival processes
- Spawn new processes
- Merge the spawned processes to the highest ranks by using intercommunicator
- Split the communicator in the original order

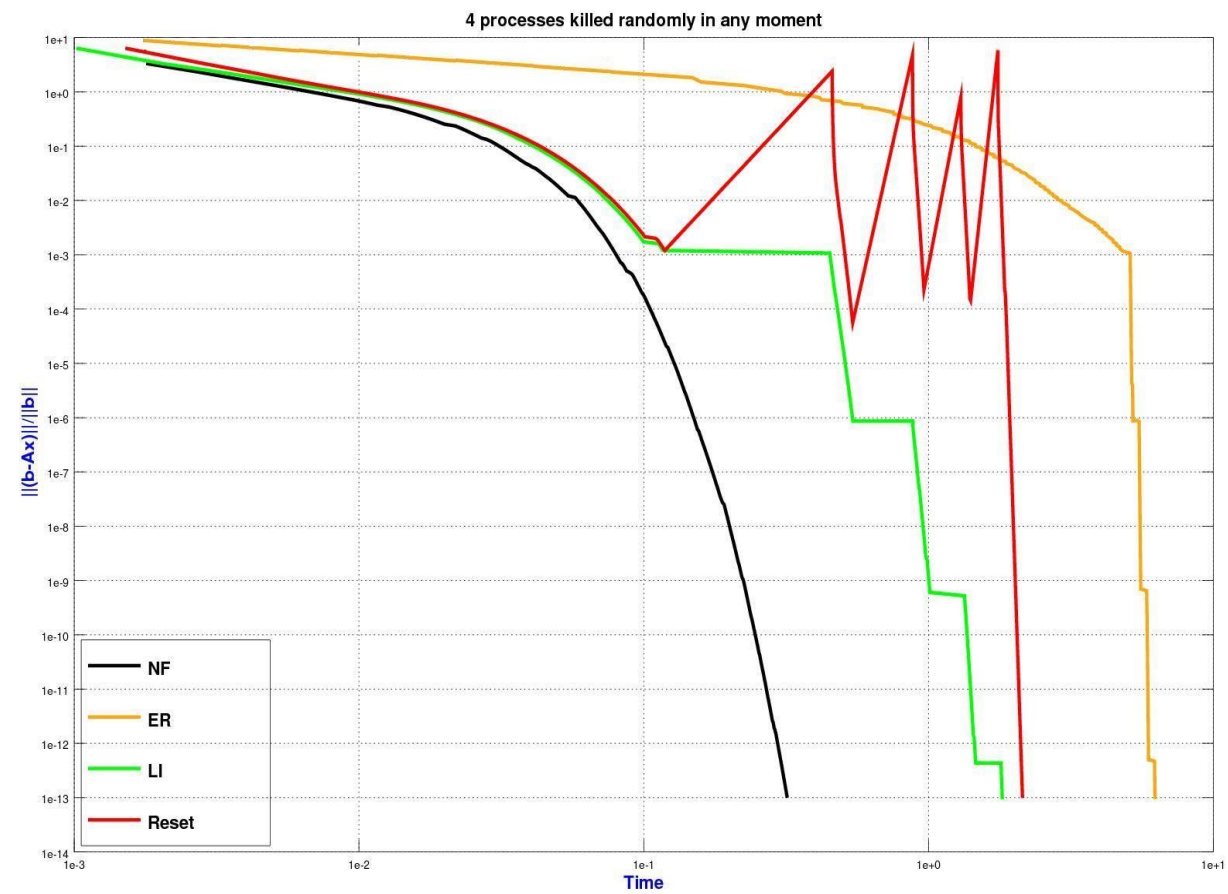


Experimental results

- 4 scenarios with 4 errors injected in different iterations
- Error injection is by Simulation (exit), or by real kills the process from the externe (Kill <pid>)
- Compare the convergence of the recovery methodes to the non-faulty method

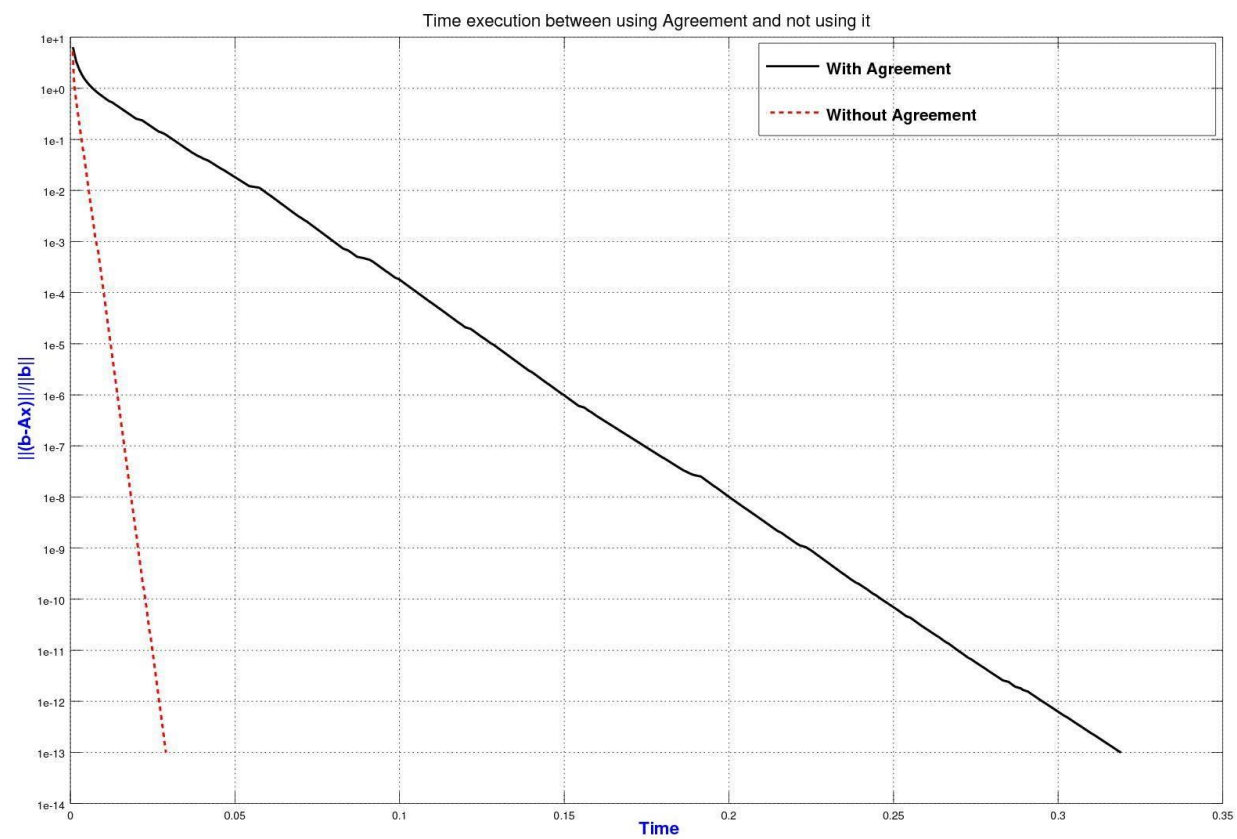
Abreviations	Signification
NF	No-faulty
ER	Checkpoint/restart
Reset	Reset
LI	Linear interpolation





The cost of Agreement

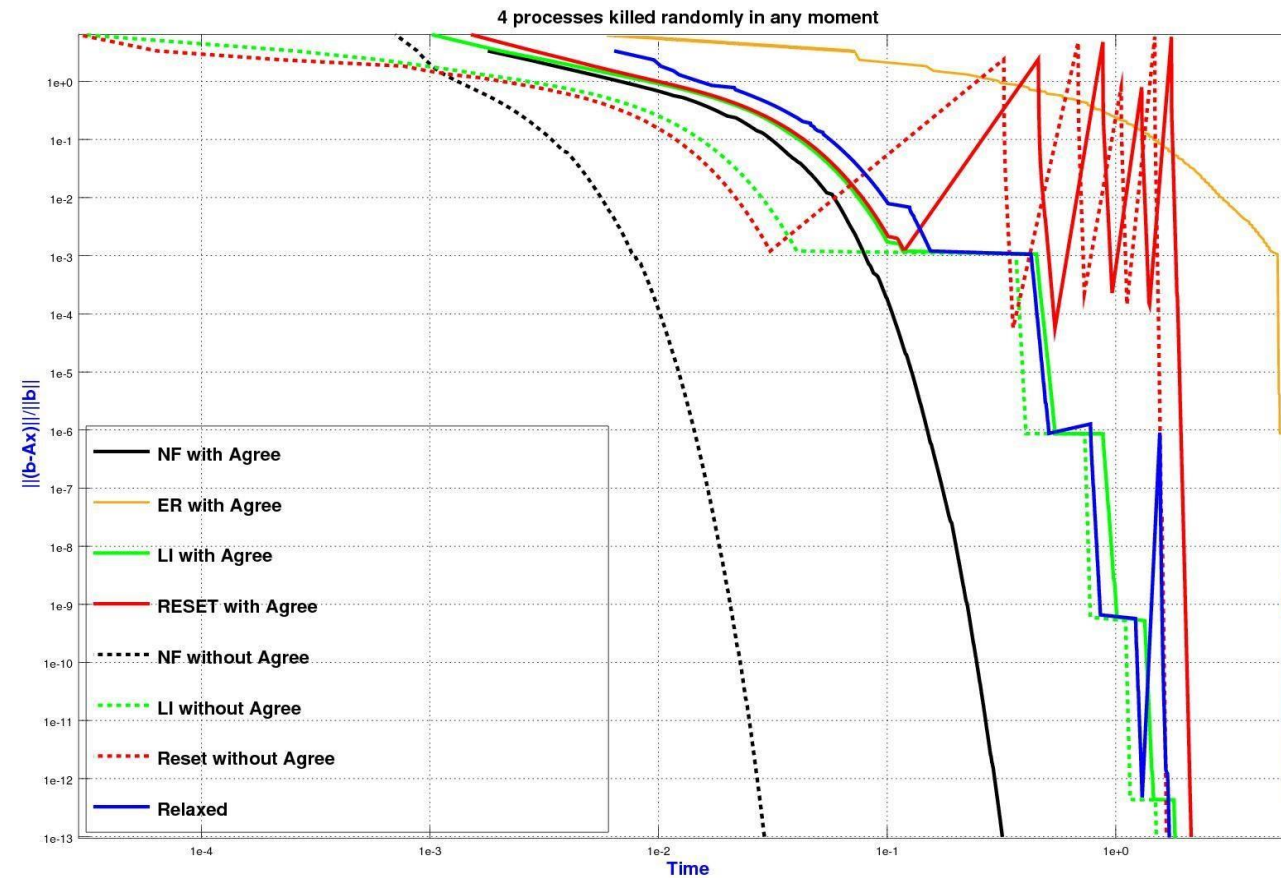
- It can be conceptualized as a failure resilient reduction on a boolean value.
 - The algorithm performs a reduction of the input values to an elected coordinator in the communicator
 - The coordinator makes decision on the output value and broadcast the value back to all the alive processes.
- MPI_Comm_Agree adds an important overhead to the computation time



Coordinated and uncoordinated iterations

- All the processes must achieve the iteration k before passing to iteration $k+1$
- It assures a synchronisation between processes.
- By taking off Agreement and barriers, processes can be either at iteration k or $k+1$, known as uncoordinated iterations
- The coordinated version can be seen as chaotic relaxation of Jacobi iteration

The computation time of recovery strategies





Conclusion

- Failures can be caused by a variety of reasons of memory, software and hardware.
- We designed and implemented toolkit using ULFM that can be directly integrated into any linear solver with any iterative method.
- Recovery strategies effectiveness depends on the type of the problem.
- For linear system solvers with iterative methods, the Linear Interpolation (LI) technique gives the best Performance-time ratio.

Conclusion


Recovery strategy	Definition	Fault tolerance supported in this project
Reset	Replace the lost data by its initial value	Multiple Faults
Checkpoint/restart	Stores the whole state and rollback if failure	Multiple Faults
Linear interpolation	Interpolate the data from the dynamic data of the survival processes	Single Fault
Relaxed	Uncoordinated version of Checkpoint/restart	Multiple Faults

- This project including explications, graphs, and other resilient examples that i did with ULFM are avaible at : https://github.com/samirouche/ft_uflm



Prospects

- Using the same ULFM communicator recovery's mechanisms in Krylov linear solvers
- Optimize the utilisation of the costly functions like `MPI_Comm_Agree`
- Push it to the ultimate conditions and see after that how numerical recovery strategies behave



Thank you for your
attention !