ULFM

Обзор функциональных возможностей

User Level Failure Mitigation

The User Level Failure Mitigation (ULFM) proposal was developed by the MPI Forum's Fault Tolerance Working Group [1].

Designing the mechanism that users would use to manage failures was built around three concepts:

1. **simplicity**, the API should be easy to understand and use in most common scenarios;
2. **flexibility**, the API should allow varied fault tolerant models to be built as external libraries and;
3. **absence of deadlock**, no MPI call (point-to-point or collective) can block indefinitely after a failure, but must either succeed or raise an MPI error.

[1] http://fault-tolerance.org/

User Level Failure Mitigation

To use this ULFM implementation, an MPI application must change the default error handler on (at least)

- **MPI_COMM_WORLD from MPI_ERRORS_ARE_FATAL**

to either

- **MPI_ERRORS_RETURN or a custom MPI Errorhandler**

User Level Failure Mitigation

Предложение по смягчению последствий на уровне пользователя (ULFM) было разработано рабочей группой MPI Forum Fault Tolerance [1].

Проектирование механизма, который пользователи будут использовать для управления отказами, строится вокруг трех концепций:

1. **простота,** API должен быть легко понят и использоваться в большинстве распространенных сценариев;
2. **гибкость,** API должен позволять создавать различные отказоустойчивые модели как внешние библиотеки;
3. **отсутствие взаимоблокировки,** никакой вызов MPI (точка-точка или коллективный) не может блокироваться неограниченно после сбоя, он должен либо корректно завершиться, либо сообщить об ошибке.

[1] http://fault-tolerance.org/

Overview

- Code Repository: https://bitbucket.org/icldistcomp/ulfm
- Bulding&Running: http://fault-tolerance.org/ulfm/ulfm-setup/
- ULFM proposal ticket: https://github.com/mpi-forum/mpi-issues/issues/20
- MPI4.0 ???
- ULFM is prototype implementation in OpenMPI

Сборка&Запуск&Тестирование

- http://fault-tolerance.org/ulfm/ulfm-setup/
- `hg clone ssh://hg@bitbucket.org/icldistcomp/ulfm`
- `./configure && make && make install`
- Сборка и установка происходит также как и OpenMPI

You will need to add one additional header to get access to the new functions, just after the include for mpi.h:

For C/C++ this looks like:

```
#include <mpi.h>
#include <mpi-ext.h>
```

Error-handler sample

```
static void verbose_errhandler(MPI_Comm* comm, int* err, ...) {
    int rank, size, len;
    char errstr[MPI_MAX_ERROR_STRING];
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Error_string( *err, errstr, &len );
    printf("Rank %d / %d: Notified of error %s\n", rank, size, errstr);
}
int main(int argc, char *argv[]) {
    int rank, size;
    MPI_Errhandler errh;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_create_errhandler(verbose_errhandler, &errh); // создаем обработчик
    MPI_Comm_set_errhandler(MPI_COMM_WORLD, errh);
    if (rank == (size-1)) { raise(SIGKILL); } // Убиваем процесс
    MPI_Barrier(MPI_COMM_WORLD);// Точка синхронизации
    printf("Rank %d / %d: Stayin' alive!\n", rank, size);
    MPI_Finalize();
    return 0;
```

Respawn sample

```
static int MPIX_Comm_replace(MPI_Comm comm, MPI_Comm *newcomm) {
        MPIX_Comm_shrink(comm, &scomm); MPI_Comm_size(scomm, &ns); MPI_Comm_size(comm, &nc);
        MPI_Comm_set_errhandler( scomm, MPI_ERRORS_RETURN );
        rc = MPI_Comm_spawn(...);
}
static int app_needs_repair(MPI_Comm comm) {
        if (comm == world) {
                worldi = (worldi + 1) % 2;
                if (MPI_COMM_NULL != world) { MPI_Comm_free(&world); }
                MPIX_Comm_replace(comm, &world);
                if (world == comm) {
                        return false; // Ok, we repaired nothing, no need to redo any work
                }
        }
        return true; // We have repaired the world, we need to reexecute
}
static void errhandler_respawn(MPI_Comm* pcomm, int* errcode, ...) {
        if (MPIX_ERR_PROC_FAILED != eclass && MPIX_ERR_REVOKED != eclass) {
                MPI_Abort(MPI_COMM_WORLD, *errcode);
        }
        MPIX_Comm_revoke(*pcomm);
        app_needs_repair(*pcomm);
}
int main( int argc, char* argv[] ) {
        MPI_Comm_create_errhandler(&errhandler_respawn, &errh);
        while(iteration < max_iterations || app_needs_repair(world)) {
                if (iteration == error_iteration && victim) raise( SIGKILL );
                rc = MPI_Bcast(array, COUNT, MPI_DOUBLE, 0, world);
        }
```

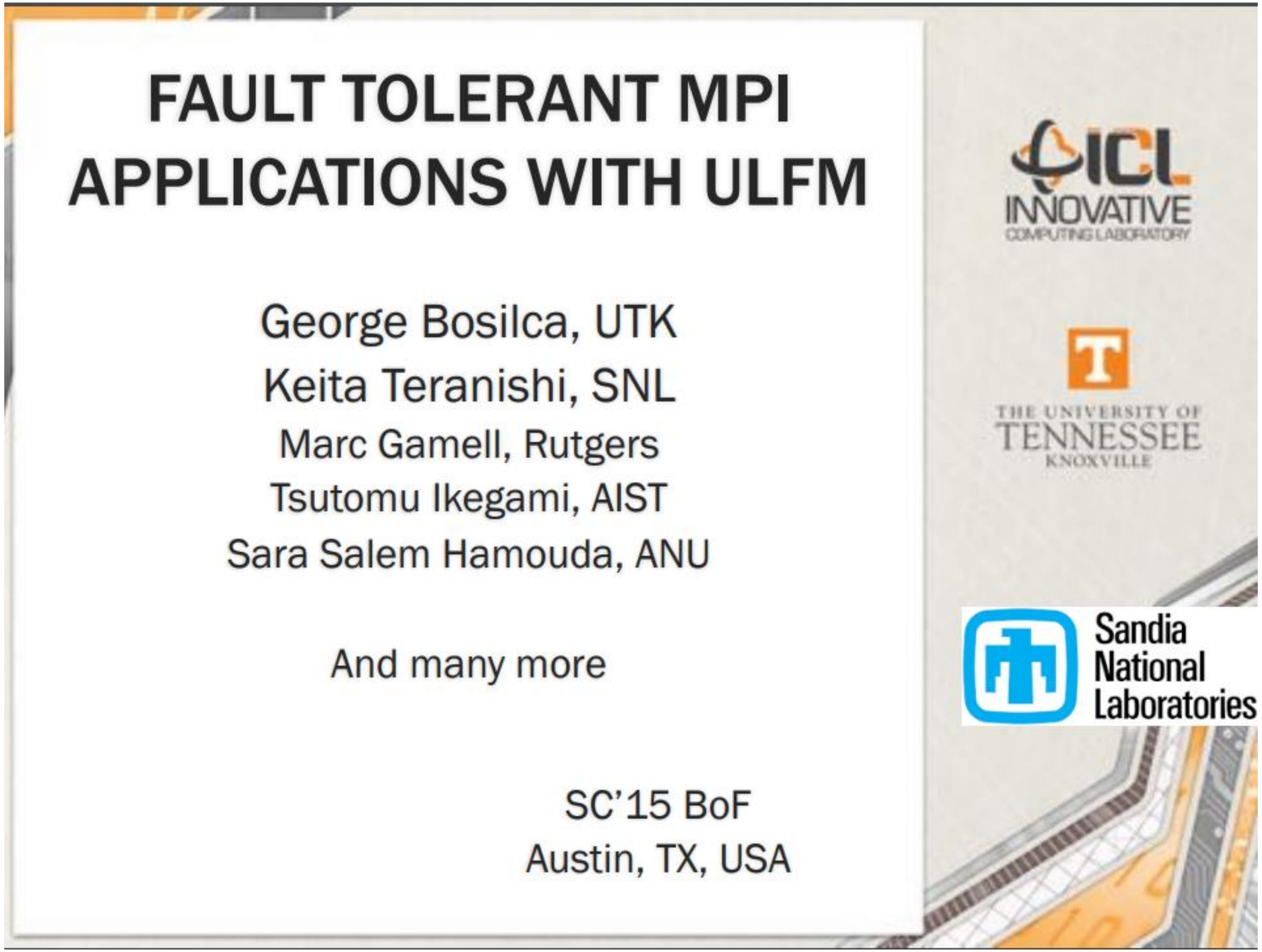
Базовые примеры

http://fault-tolerance.org/ulfm/usage-guide/

- **Master/Worker -** The example below presents a master code that handles failures by ignoring failed processes and resubmitting requests. It demonstrates the different failure cases that may occur when posting receptions from MPI_ANY_SOURCE as discussed in the advice to users in the proposal.
- **Iterative Refinement -** The example below demonstrates a method of fault-tolerance to detect and handle failures. At each iteration, the algorithm checks the return code of the MPI_ALLREDUCE. If the return code indicates a process failure for at least one process, the algorithm revokes the communicator, agrees on the presence of failures, and later shrinks it to create a new communicator.

```c
int master(void) {
    MPI_Comm_set_errhandler(comm, MPI_ERRORS_RETURN);
    MPI_Comm_size(comm, &size);
    // ... submit the initial work requests ...
    MPI_Irecv(buffer, 1, MPI_INT, MPI_ANY_SOURCE, tag, comm, &req);
    // Progress engine: Get answers, send new requests,
    // and handle process failures
    while((active_workers > 0) && work_available) {
        rc = MPI_Wait(&req, &status); // ТОЧКА СИНХРОНИЗАЦИИ!
        if((MPI_ERR_PROC_FAILED == rc) || (MPI_ERR_PENDING == rc)) {
            MPI_Comm_failure_ack(comm); MPI_Comm_failure_get_acked(comm, &g);
            MPI_Group_size(g, &gsize);
            // ... find the lost work and requeue it ...
            active_workers = size - gsize - 1;
            MPI_Group_free(&g);
            // repost the request if it matched the failed process
            if(rc == MPI_ERR_PROC_FAILED) {
                MPI_Irecv(buffer, 1, MPI_INT, MPI_ANY_SOURCE, tag, comm, &req);
            }
            continue;
        }
        // ... process the answer and update work_available ...
        MPI_Irecv(buffer, 1, MPI_INT, MPI_ANY_SOURCE, tag, comm, &req);
     }
     // ... cancel request and cleanup ...
}
```

http://fault-tolerance.org/downloads/ulfm-bof-sc15/ULFM%20Bof%20SC15.pdf



Credits: ETH Zurich

ULFM-based Applications

- **ORNL**: Molecular Dynamic simulation, C/R in memory with Shrink
- **UAB**: transactional FT programming model
- **Tsukuba**: Phalanx Master-worker framework
- **Georgia University**: Wang Landau Polymer Freezing and Collapse, localized subdomain C/R restart
- **Sandia, INRIA, Cray:** PDE sparse solver
- **Cray**: CREST miniapps, PDE solver Schwartz, PPStee (Mesh, automotive), HemeLB (Lattice Boltzmann)
- **ETH Zurich**: Monte-Carlo, on failure the global communicator (that contains spares) is shrunk, ranks reordered to recreate the same domain decomposition
- Programming models: resilient X10

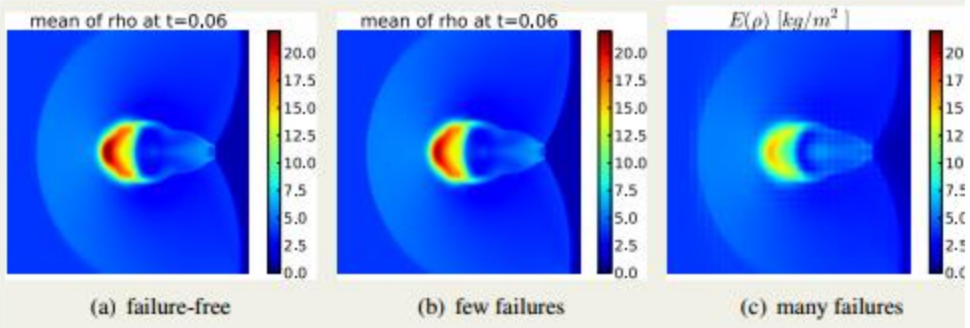
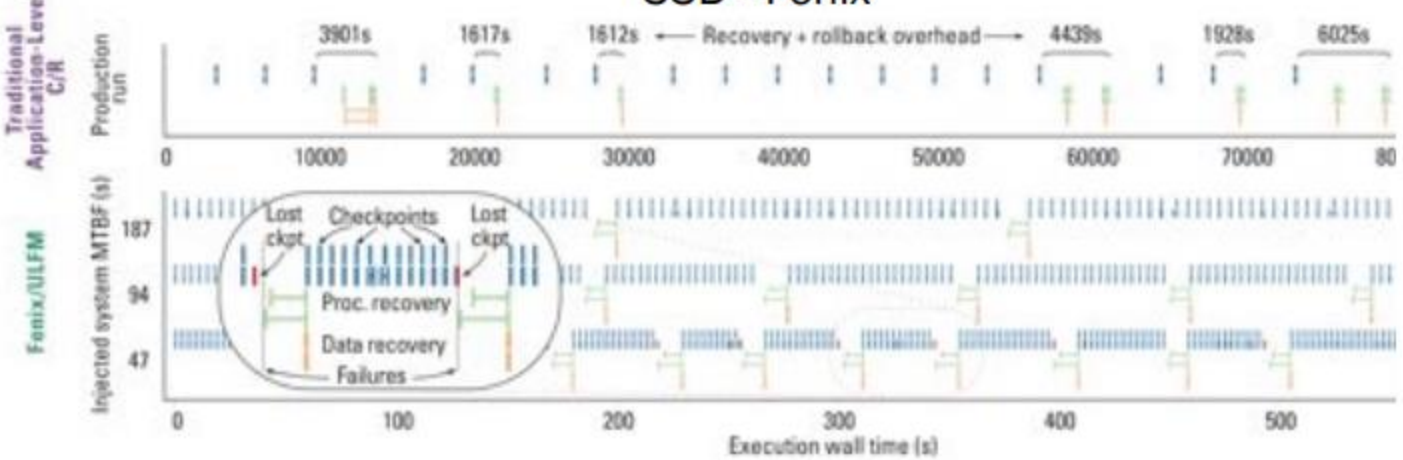


Figure 5. Results of the FT-MLMC implementation for three different failure scenarios.

FRAMEWORKS USING ULFM

LFLR, FENIX, FTLA, Falanx

S3D - Fenix



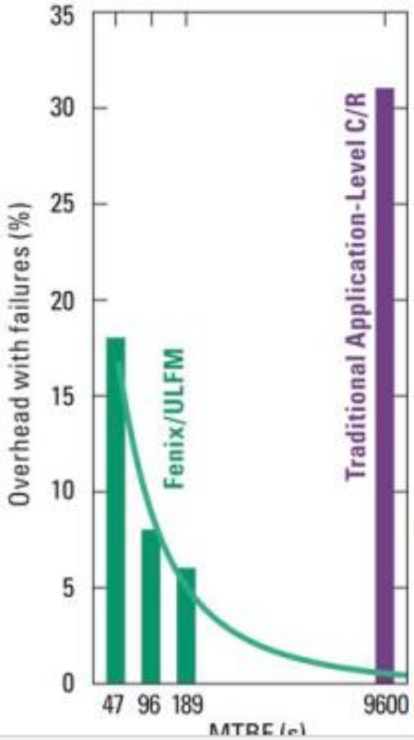
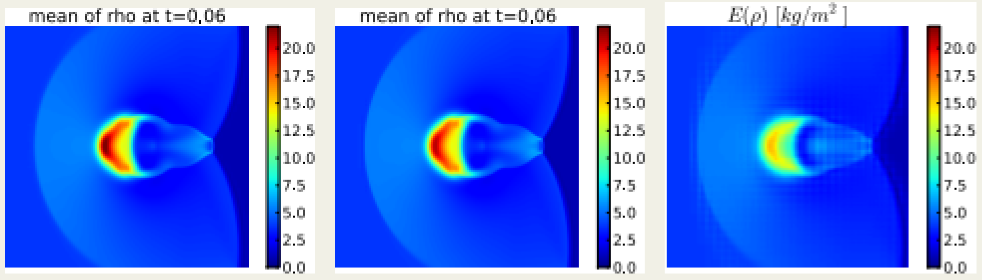


Image courtesy of the authors, M.Gamell, D.Katz, H.Kolla, J.Chen, S.Klasky, and M.P
Exploring automatic, online failure recovery for scientific applications at extreme s
In Proceedings of SC '14

http://fault-tolerance.org/wp-content/uploads/2014/05/20140527-ULFM-users.pdf

User activities

- ORNL: Molecular Dynamic simulation
  - Employs coordinated user-level C/R, in place restart with Shrink
- UAB: transactional FT programming model
- Tsukuba: Phalanx Master-worker framework
- Georgia University: Wang Landau Polymer Freezing and Collapse
  - Employs two-level communication scheme with group checkpoints
  - Upon failure, the tightly coupled group restarts from checkpoint, the other distant groups continue undisturbed
- Sandia: PDE sparse solver
- INRIA: Sparse PDE solver
- Cray: CREST miniapps, PDE solver Schwartz, PPStee (Mesh, automotive), HemeLB (Lattice Boltzmann)
- UTK: FTLA (dense Linear Algebra)
  - Employs ABFT
  - FTQR returns an error to the app, App calls new BLACS repair constructs (spawn new processes with MPI_COMM_SPAWN), and re-enters FTQR to resume (ABFT recovery embedded)
- ETH Zurich: Monte-Carlo
  - Upon failure, shrink the global communicator (that contains spares) to recreate the same domain decomposition, restart MC with same rank mapping as before



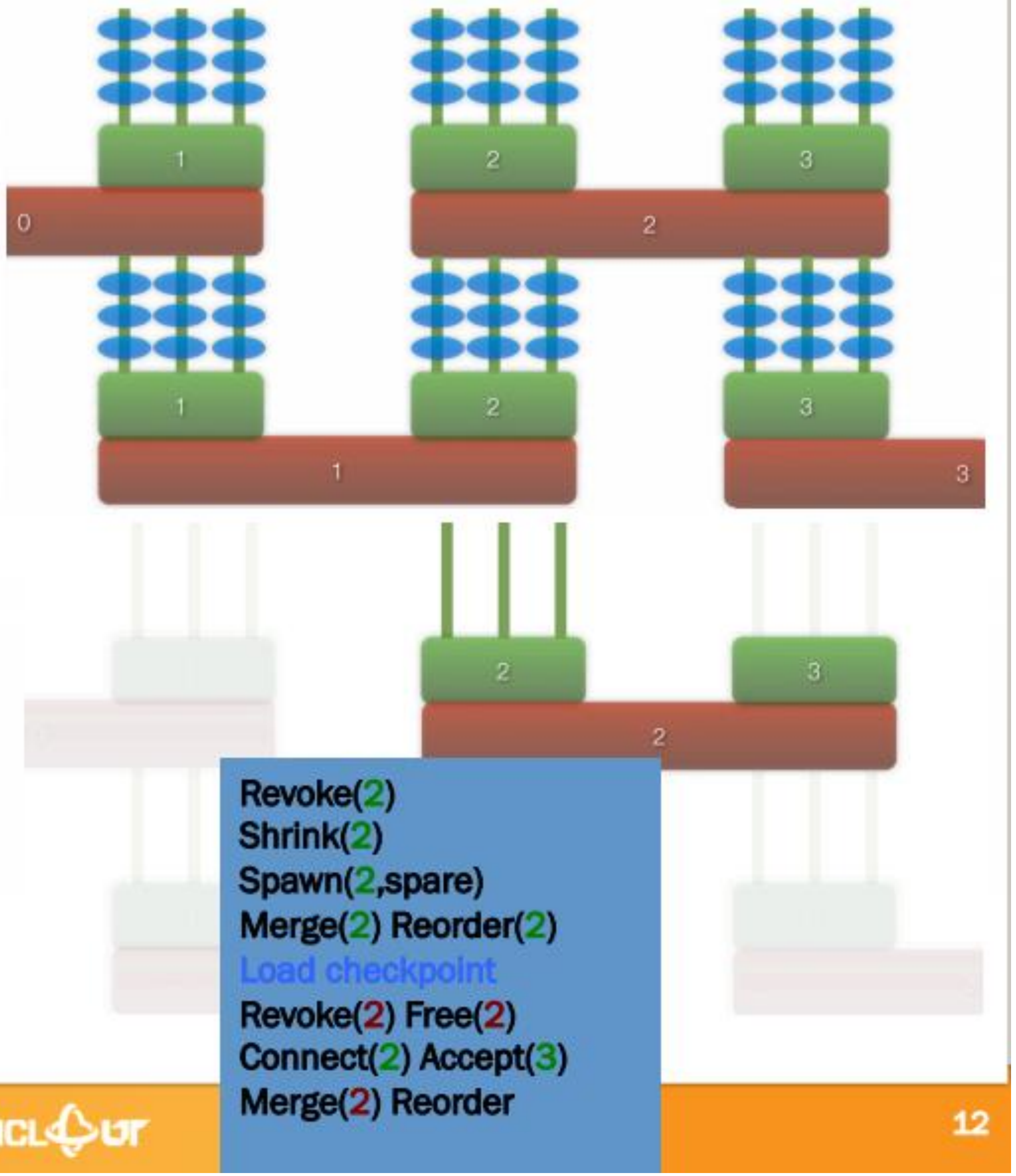
(a) failure-free (b) few failures (c) many failures

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

Credits: ETH Zurich

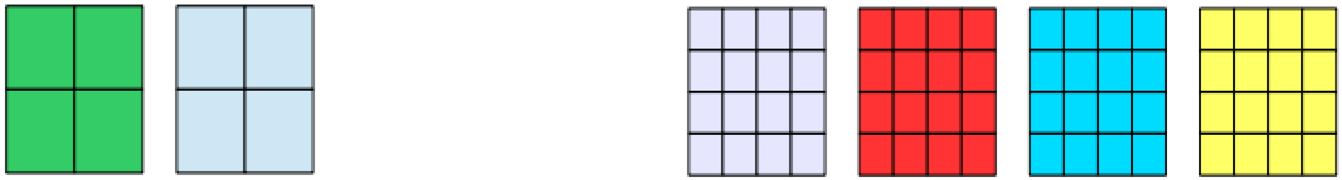
U-GA: Wang-Landau polymer Freeze

- Long independent computation on each processor
  - Dataset protected by small, cheap checkpoints (stored on neighbors)
- Periodically, an AllReduce on the communicator of the Energy window
- Immediately after, a Scatter and many pt2pt on the communicator linking neighboring energy windows



ETH-Zurich: Monte-Carlo PDE

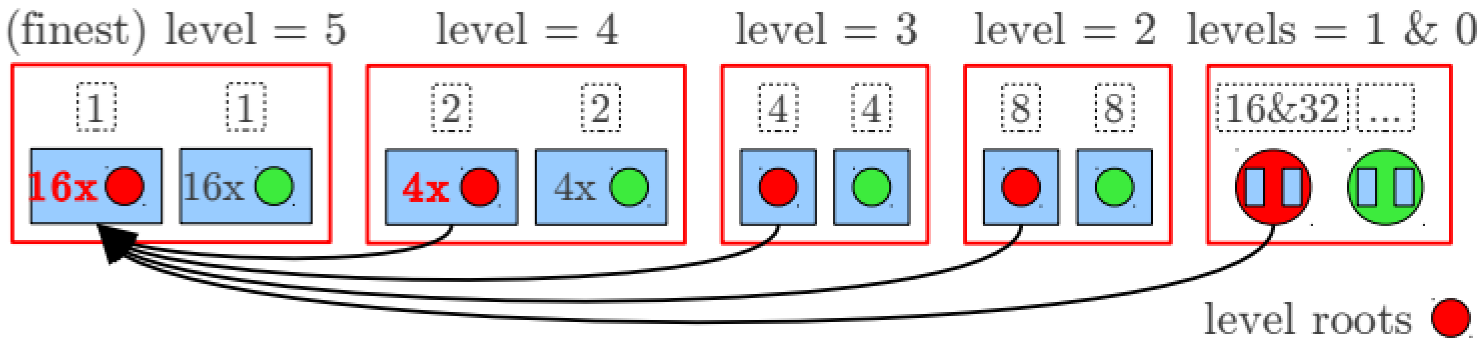
- X is the solution to a stochastic PDE
- Each sample X^i is computed with a FVM solver
- MC error is determined by
  - stochastic error (depends on M)
  - discretization error (depends on the mesh-width h)
- A more accurate MC approximation requires more samples M and a finer mesh h



Fault Tolerant Monte Carlo:

- The number of samples M turns into a random variable $\hat{M}$
- $\|\mathbb{E}[X] - E_{\hat{M}}[X]\| \leq \mathbb{E}\left[\frac{1}{\sqrt{\hat{M}}}\right] \|X\|$

ETH-Zurich: Monte-Carlo PDE



- Try to collect the mean as in fault-free ALSVID-UQ
- Call `MPI_BARRIER` on `MPI_COMM_WORLD` at the end to discover failed processes
- non-uniform success of `MPI_BARRIER`: `MPI_BARRIER` is followed by `MPI_COMM_AGREE`
- In case of failure: (Re)assign the level roots and repeat the collection of the means