Parallel Computing: Accelerating Computational Science and Engineering (CSE)
M. Bader et al. (Eds.)
IOS Press, 2014

doi:10.3233/978-1-61499-381-0-471

A fault tolerant implementation of Multi-Level Monte Carlo methods

Stefan Pauli [a,b,1,2], Manuel Kohler [a,3], Peter Arbenz [a,4]
[a] *ETH Zurich, Computer Science Department, 8092 Zurich, Switzerland*
[b] *ETH Zurich, Seminar for Applied Mathematics, 8092 Zurich, Switzerland*

Abstract. The theory behind fault tolerant multi-level Monte Carlo (FT-MLMC) methods was recently developed and tested. These tests were made without a real fault tolerant implementation. We implemented an MPI-parallelized fault tolerant MLMC version of an existing parallel MLMC code (ALSVID-UQ). It is based on the User Level Failure Mitigation, a fault tolerant extension of MPI. We confirm our FT-MLMC theory by means of simulations of the two-dimensional stochastic Euler equations of gas dynamics.

Keywords. Multi-level Monte Carlo, fault tolerance, MPI, ULFM

Introduction

In large scale simulations on emerging massively parallel computing platforms processor failures at runtime are inevitable [1] and occur, in fact, with increasing frequency as the number of processors increases. This trend will lead to a situation where standard checkpoint/restart mechanisms will no longer work properly. This motivated us to propose a fault tolerant Monte Carlo (FT-MC) and a fault tolerant multi-level Monte Carlo (FT-MLMC) method neither relying on checkpoint/restart nor on recomputation of samples [2]. We proposed to incorporate all samples unaffected by failures in the computation of the final result, and simply ignore samples affected by failures. Failures therefore do not lead to an overhead, but potentially lower the quality of the results. Silent (undetected) errors were not covered by this theory.

In this paper we present an implementation of the FT-MLMC method. This implementation tolerates failures of several cores of a high performance computer. We use the User Level Failure Mitigation (ULFM) [3], a fault tolerant extension of MPI, to achieve this behavior. In order to test the program a failure generator aborts MPI processes after randomly chosen execution times.

First, in Section 1 we introduce the concept of MLMC and the theoretical properties of its fault tolerant version FT-MLMC. Then, we describe the implementation in Section 2. In Section 3 we present the FT-MLMC error with respect to simulated fail-

[1] Corresponding author. E-mail: stefan.pauli@inf.ethz.ch
[2] The work of this author has been funded by the ETH interdisciplinary research grant CH1-03 10-1.
[3] E-mail: manukohl@ethz.ch
[4] E-mail: arbenz@inf.ethz.ch

ure rates. The execution time of the FT-MLMC method is compared with the standard failure-free MLMC method and the influence of the failures is quantified. In Section 4 we discuss and analyse the obtained results. Finally, we draw our conclusions in Section 5.

1. FT-MLMC

The convergence rate with respect to the computational work of Monte Carlo (MC) methods, applied to estimate the expected solution to a partial differential equation with random input, is often suboptimal. The accuracy of the MC method is determined on the one hand by the number of samples used and on the other hand by the discretization error accepted in the computation of each sample (for instance using a finite volume method). To increase the accuracy of the MC solution, it is not sufficient to simply use more samples, additionally each sample has to be computed with a lower discretization error. This results in higher computational costs per sample, as for instance by using a finer mesh. This is illustrated in Fig. 1, where two MC simulations are shown, on the left an inaccurate, and on the right a more accurate one. The error of MC methods normally converges as

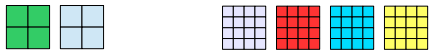


Figure 1. On the left a inaccurate MC simulation with 2 samples inaccurately computed on a coarse mesh. On the right a more accurate MC simulation with more samples computed on a finer mesh.

$1/\sqrt{M}$, where M is the number of samples. If all samples were equally expensive the error versus work convergence rate would be $1/\sqrt{\text{work}}$. But, increasing accuracy demands more accurately computed samples, which comes with increasing computation costs per sample. This explains why the MC convergence versus work is worse than $1/\sqrt{\text{work}}$. If applicable, Multi-level Monte Carlo (MLMC) methods may reach this optimal $1/\sqrt{\text{work}}$ convergence rate.

In MLMC methods [4,5,6,7] not all samples are computed on the finest mesh. In fact, most samples are computed on coarser meshes with a larger discretization error. We specify a hierarchy of discretization levels with the corresponding mesh width, $h_\ell < h_{\ell-1}$, from the coarsest level $\ell = 0$ to the finest $\ell = L$. We denote by $X_{h_\ell} - X_{h_{\ell-1}}$ a sample on level ℓ, where X_{h_ℓ} is computed with a mesh width h_ℓ. Using a telescopic sum the MLMC approximation is defined as

$$E[X_{h_L}] = E_{M_0}[X_{h_0}] + \sum_{\ell=1}^{L} E_{M_\ell}[X_{h_\ell} - X_{h_{\ell-1}}].$$

Here, $E_{M_\ell}[\cdot]$ denotes the mean estimated with MC using M_ℓ samples. First, the mean of the coarsest discretization level $\ell = 0$ is computed. Then, the difference from each consecutive discretization level to the next is approximated and added. The MLMC approximation is illustrated and compared to an MC method in Fig. 2, where every color symbolizes a sample with equal input parameters and the discretization level is illustrated by the underlying mesh.

Two solutions $X_{h_\ell}, X_{h_{\ell-1}}$ computed on consecutive discretization levels are similar. The MLMC method exploits that the variance of the difference $X_{h_\ell} - X_{h_{\ell-1}}$ decreases as ℓ

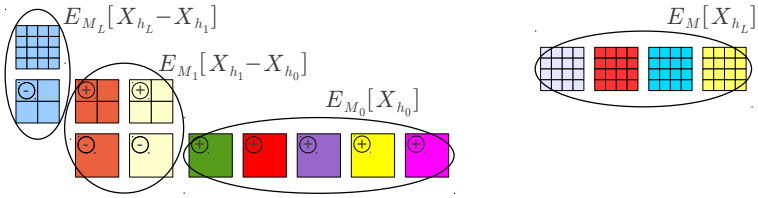


Figure 2. The idea of MLMC is illustrated on the left and compared to the MC method on the right.

increases. The error of the MC estimate is determined by this variance and by the number of samples M_ℓ. As for large ℓ (fine mesh, and hence expensive) the variance is small, it is possible to use fewer samples on fine levels, such that $M_0 > M_1 > \ldots > M_L$. More precisely, the MLMC error is bounded by

$$\begin{aligned} \|\mathbb{E}[X] - E[X_{h_L}]\|_{L^2(\Omega;L^1)} \leq \|\mathbb{E}[X] - \mathbb{E}[X_{h_L}]\|_{L^1} + M_0^{-1/2}\|X_{h_0}\|_{L^2(\Omega;L^1)} \\ + \sum_{\ell=1}^{L} M_\ell^{-1/2}\|X_{h_\ell} - X_{h_{\ell-1}}\|_{L^2(\Omega;L^1)}. \end{aligned} \tag{1}$$

For many problems a clever choice of M_ℓ leads to the optimal MLMC convergence rate of error versus $1/\sqrt{\text{work}}$ [4,5,6,7].

In [2] we proposed a fault tolerant MLMC (FT-MLMC) algorithm, which uses only the surviving samples (unaffected by failures) to compute the MLMC estimate. The number of surviving samples is not a fixed number, but a random variable $\hat{M}_\ell$ from the failure probability space Ω'. The resulting error is bounded by

$$\begin{aligned} \|\mathbb{E}[X] - E[X_{h_L}]\|_{L^1(\Omega';L^2(\Omega;L^1))} \leq \|\mathbb{E}[X] - \mathbb{E}[X_{h_L}]\|_{L^1(\Omega';L^1)} \\ + \mathbb{E}[\min(\hat{M}_0^{-1/2}, 1)]\ \|X_{h_0}\|_{L^2(\Omega;L^1)} + \sum_{\ell=1}^{L} \mathbb{E}[\min(\hat{M}_\ell^{-1/2}, 1)]\ \|X_{h_\ell} - X_{h_{\ell-1}}\|_{L^2(\Omega;L^1)}, \end{aligned}$$

an error bound very similar to the failure-free MLMC method. Compared with (1), $M_\ell^{-1/2}$ is replaced by the expected value $\mathbb{E}[\min(\hat{M}_\ell^{-1/2}, 1)]$ (over the failure probability space Ω'). The minimum covers the case where all samples of a level ℓ are lost. This introduces a significant error in the FT-MLMC estimate [2]. Note that the coarser the level, the larger the error, which makes it fatal to lose coarse levels.

2. Implementation

We implemented FT-MLMC using ALSVID-UQ [8], an existing MPI-parallelized code designed for uncertainty quantification in partial differential equations using MLMC simulations. In order to achieve fault tolerance with MPI, we chose the User Level Failure Mitigation (ULFM) [3].

2.1. ULFM

Originally, ULFM [3] was proposed as a process fault tolerance extension in the MPI-3.0 standard. However, it was removed from the standard before its release. It consists of

a minimal set of changes, necessary for libraries and applications, to incorporate fault tolerance. ULFM is implemented in the development branch 1.7ft of Open MPI (We use git rev. 2d7175b).

In ULFM, process failures are reported using the return code of MPI communication routines. A point-to-point communication routine returns with success or it (eventually) reports the failure of the partner process. In collective communication, the result might be non-uniform, i.e., some processes report process failures, while others successfully terminate the current communication. `MPI_COMM_REVOKE` can be used to explicitly propagate knowledge about failures and prohibit any further communication on the given communicator by setting the communicator in the revoked state. Using `MPI_COMM_SHRINK`, a new communicator is created containing all surviving processes of a revoked communicator. Additionally, ULFM provides a consensus protocol by the `MPI_COMM_AGREE` routine.

2.2. *Parallelization*

The parallelization of FT-MLMC follows the approach made in ALSVID-UQ [9]. As illustrated in Fig. 3, levels (red) are executed in parallel, like samples of the same level (blue) and subdomains, when domain decomposition is applied to large samples of fine levels. The load of each process can be balanced statically as samples of coarse levels require a known fraction of the execution time of a sample of the finest level [9]. Samples

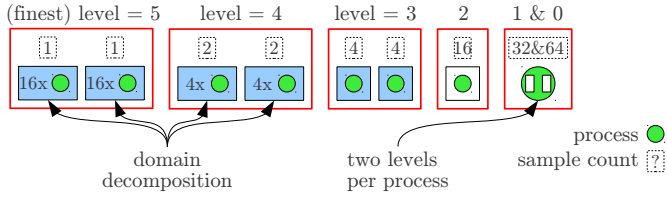


Figure 3. Parallel distribution of work in ALSVID-UQ [9].

of very coarse levels are computed by a single process. In the actual implementation of ALSVID-UQ multiple levels (e.g., 0 and 1 in Fig. 3) can be dealt with by one process. If one of these processes fails, all samples of the respective level are lost, leaving us with a large error. In order to avoid this, in FT-MLMC, we assign at least two processes to a level such that at least two processes have to fail until all samples of a level are lost. This is illustrated in Fig. 4, where levels 0 to 2 are distributed among 2 processes. Note that processes dealing with the coarsest levels are usually underloaded. However, the performance drop is marginal for large runs.

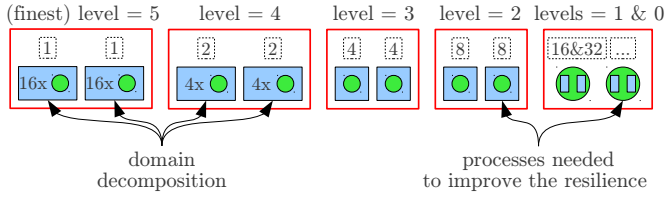


Figure 4. Parallel distribution of work in FT-MLMC with improved failure resilience.

2.3. Communication

The failure-free ALSVID-UQ consists of three communication phases. The first phase concerns large samples, where information is exchanged across the interfaces of subdomains during the computation. In the second phase, the mean is calculated on every level using `MPI_REDUCE` on the level communicator, i.e., a communicator connecting domains on the same level or, for large samples, corresponding subdomains on the same level. The means are stored in 'level roots'. In the third phase, all level roots send their means by point-to-point communication to the level roots of the finest level, where they are aggregated and saved to disk.

We have kept the communication pattern of ALSVID-UQ and supplemented it with fault tolerance. In the first phase, a process stops computing as soon as it discovers the failure of one of the processes dealing with another subdomain of the same sample. However, it continues participating in communications. In the second phase, we use `MPI_ALLREDUCE` instead of `MPI_REDUCE`, such that each process owns the mean of its level. This way, any process can play the role of the level root. In case of a failure, the level communicator is recreated and `MPI_ALLREDUCE` is repeated with the remaining processes. If all processes of a level communicator fail, the level is lost. In rare cases, a process failure can cause non-uniform success of `MPI_ALLREDUCE`. Hence, `MPI_ALLREDUCE` is followed by `MPI_COMM_AGREE` to ensure that all processes together either repeat `MPI_ALLREDUCE` as just described or continue with the third phase.

In the third phase, all processes attempt to aggregate the mean values of the different levels and store the result to disk. Before any of the involved processes finalizes MPI, they must check whether this procedure has completed without failure. To this end, `MPI_BARRIER` is called on `MPI_COMM_WORLD` to discover failed processes. If `MPI_BARRIER` indicates a failure, then each level communicator has to (re)assign its level root, maybe after a recreation of the communicator. Additionally, the finest living level has to be determined. Afterwards, the attempt to aggregate can be resumed. It has to be paid attention if `MPI_BARRIER` indicates success. As the success of the `MPI_BARRIER` is non-uniform, `MPI_COMM_AGREE` must be called to enforce globally consistent information. If this means success, then all processes know that the MLMC result is safely stored to disk such that they can terminate.

In the described method, samples are communicated to other processes only after all samples of a level are computed. This is why we call this the "late save" strategy. To reduce the vulnerability of FT-MLMC to failures, the sum of the local samples may be sent at periodic time intervals to all processes of the level communicator, where it is added to their local sum. In case of failure the intermediate results are not lost. We call this the "intermediate save" strategy. In our implementation this is achieved by repeatedly applying the method used to calculate the mean on every level.

2.4. Failure generation

In order to test the implementation and to compare the results with our theory we included a failure generator, which terminates MPI processes randomly with a given distribution. We use a Weibull distribution [10] to model the time between two consecutive failures on one core, where we assume that core failures are statistically independent. Based on a study of Schroeder and Gibson [11], we use a Weibull shape parame-

ter $k = 0.5$, and vary the mean time between failure (MTBF) with the Weibull scale parameter λ. The mean of the Weibull distribution represents then the MTBF for a core. Dividing this result by the number of cores involved yields the MTBF of all the cores.

When a simulation is started, we don't know when the involved cores failed last. Therefore, we are not only interested in the time between two failures but also in the distribution of the time from the start of the computation until the first failure of a core occurs. In [2] we describe an algorithm to draw a realization of this distribution, using findings in [12,13].

The failure generator is started in the initial phase of the program with given Weibull parameters λ and k. It computes a realization of the first failure time of each MPI process by drawing a realization of the described distribution. For the given failure times a timed asynchronous interrupt is set, which kills the MPI process using the exit system call. As a consequence, failures can happen at any time, during computation, communication or while ULFM is recovering from previous failures.

3. Experiments

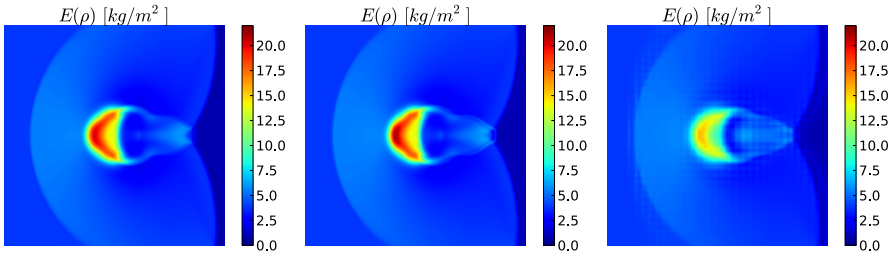
We demonstrate the ability of our fault tolerant version of ALSVID-UQ by solving the two-dimensional stochastic Euler equations of gas dynamics. In these equations a solution X_h consists of the computed density ρ_h, the velocity field $\mathbf{u}_h$ and the pressure p_h. We compute the mean (at time $t = 0.06$) of the so-called cloud-shock interaction problem with 8 sources of uncertainty in the initial condition (for details see [14, §8.1.2]). A first order finite volume method is used to compute the individual samples.

We compute our results on Brutus[5], a large compute cluster at ETH Zurich. We use nodes with four 12-core AMD Opteron 6174 CPUs and 64 GB of RAM. Due to incompatibility of Open MPI 1.7ft with the batch system of Brutus, we are currently not able to use ULFM on multiple nodes. Therefore, we show results which are computed on a single 48-core node, hence we use at most 48 MPI processes. We emphasize however that our implementation would work without any changes on multiple nodes, once the incompatibilities are resolved.

The simulations presented use the FT-MLMC configuration in Fig. 4. As shown there we use 6 levels, 46 processes, domain decomposition on levels 4 and 5 and compute a total of 126 samples. The mesh widths used are $h_\ell = 2^{-(4+\ell)}$, $0 \leq \ell \leq 5$, which means that the finest level is computed with $h_L = h_5 = 2^{-9}$. In the "intermediate save" strategy the locally computed intermediate result is sent to all processes of the same level up to 4 times. On level 5 only two samples are computed, each on 16 cores. Hence, no intermediate result is available on this level. There, the "intermediate save" strategy does not improve the failure resilience. The "intermediate save" strategy improves the resilience on levels 0 to 3, where more than 4 samples are computed per process and slightly improves it on level 4.

In Fig. 5 we show three different results, the mean of the density ρ at $t = 0.06$s, obtained by FT-MLMC. The result in Fig. 5(a) is computed failure-free. In Fig. 5(b) a result is shown, where 3 out of 46 processes were killed. Figure 5(c) shows the result of an FT-MLMC run, where 9 out of 46 processes were killed, among them both processes

[5] http://brutuswiki.ethz.ch/brutus/Brutus_cluster



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

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

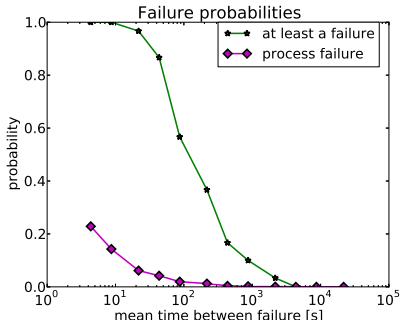
dealing with level 2, such that all samples of this level are missing. The deterioration of the result is obvious.

In Fig. 6 several quantities of the FT-MLMC method for different MTBFs are presented. All quantities are averages over 30 FT-MLMC runs. They are discussed in the next section. Figure 6(a) presents two measurements for the "intermediate save" strategy. (The results for the "late save" strategy are similar.) First, we show the percentage of processes failed during the computation. Second, the "at least a failure" probability is shown, which measures the fraction of FT-MLMC runs, that experience at least one failure. (All other FT-MLMC runs are failure-free.) Remember that standard MPI crashes if a failure occurs.

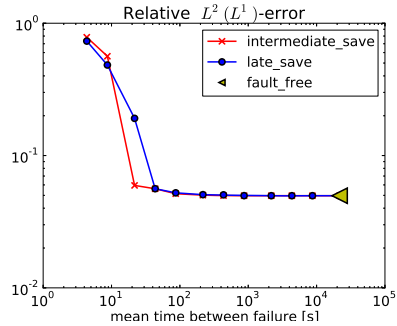
The measurement of the FT-MLMC error versus MTBF is shown in Fig. 6(b). Using a MLMC reference solution $E[X_{\text{ref}}]$ with $L = 8$, $M_L = 8$ and $h_L = 2^{-11}$, the absolute FT-MLMC error is measured as $\sqrt{E_{30}[\|E[X_{h_L}] - E[X_{\text{ref}}]\|_{L^1}^2]}$. The relative error of the failure-free ALSVID-UQ is shown at MTBF $= 2{\cdot}10^4$ s, where the fault tolerant strategies ("intermediate save" and "late save") are of the same quality. For MTBF in the range of $2 \cdot 10^4$ s downto 200 s the error remains rather constant. Then it starts to slightly grow. It "explodes" at MTBF < 40 s for the "late save" strategy and at MTBF < 20 s for the "intermediate save" strategy.

Figure 6(c) shows the measured wall-clock run-time for the two fault tolerant strategies and the failure-free run. Between the two fault tolerant versions no significant difference is measured. At MTBF $= 2{\cdot}10^4$ s we can see the small overhead (around 5%) of both fault tolerant runs compared to the standard failure-free ALSVID-UQ implementation. For the fault tolerant versions the run-time remains approximately constant for MTBF > 100 s. Then, the run-time decreases.

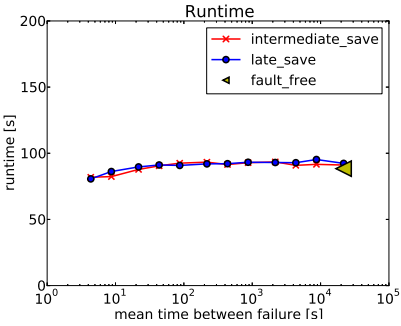
Figure 6(d) presents two measurements for the "intermediate save" strategy. (The results for the "late save" strategy are similar.) The first measurement "all samples failed" shows how often no samples at all could be computed, since too many processes have failed. Then no FT-MLMC result is computed, such that these runs are ignored in the error computation of Fig. 6(b). The same holds for runs which crashed (indicated by "program crashed").



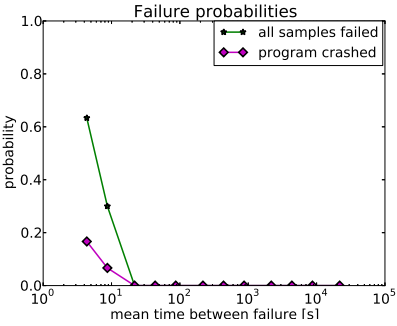
(a) failure probabilities versus MTBF



(b) the measured relative error versus MTBF



(c) wall-clock run-time versus MTBF



(d) failure probabilities versus MTBF

Figure 6. Statistics for the FT-MLMC implementation.

4. Discussion

The correlation between process failure probability and MTBF is clearly visible. However, we use the "at least a failure" probability to determine an upper MTBF rate for which fault tolerance is useful. For this purpose the "at least a failure" probability is used. At MTBF $= 2 \cdot 10^4$ s our MLMC run is rarely hit by a failure, hence fault tolerance is dispensable. However, for MTBF $< 10^3$ s, the fault tolerance pays off. Without it, more than 10% of all runs would terminate without a result, even though the process failure probability is still small.

For very small MTBFs ($\lesssim 20$ s) our FT-MLMC method does no longer perform well. This has the following two reasons:

1. If so many processes fail that no samples can be computed anymore, then no result is obtained. This happens, if all processes handling the coarsest levels fail and one process per sample of the finer levels. This effect starts to play a role for MTBF < 20 s, see Fig. 6(d). In our example (recall Fig. 4) at least 10 processes have to fail, namely all processes of levels 0 to 3 and at least one for every domain in levels 4 and 5.
2. Samples of the finer level are very vulnerable, as they are treated by many processes. If the MTBF is sufficiently small, then also processes handling the coarsest levels are frequently hit by failures and the error can get unacceptably large.

In our measurements this starts at MTBF $\approx 40\,\mathrm{s}$ for the "late save" strategy and at MTBF $\approx 20\,\mathrm{s}$ for the "intermediate save" strategy, see Fig. 6(b).
The effect is depicted in Fig. 5(c), where an FT-MLMC run is shown with a very large error.

We can now define a region, where our FT-MLMC method is helpful, i.e., where the fault tolerance is needed. In this region the "at least a failure" risk is high. However, we can still expect sufficiently many samples to complete such that the error remains small. For the "late save" strategy this region is between $40\,\mathrm{s} < \mathrm{MTBF} < 10^3\,\mathrm{s}$ and for the "intermediate save" strategy between $20\,\mathrm{s} < \mathrm{MTBF} < 10^3\,\mathrm{s}$.

The crashes we observed were mostly due to a consensus protocol failure [6], something which should not happen in ULFM. But keep in mind, that we are using a development implementation. Nevertheless, we observe that these problems only show up for $\mathrm{MTBF} < 20\,\mathrm{s}$, where the relative error is already large, and hence our fault tolerance approach is not suitable. Therefore, the implementation of ULFM is stable enough for these runs.

As described in Section 2 some additional commands are added to ALSVID-UQ, in order to make it fault tolerant. This has an effect on the wall-clock run-time, independent of the appearance of failures. Additionally, the standard (failure-free) MPI used on our cluster Brutus might be slightly faster than the implementation of ULFM we used. The combination of these effects is measurable and results in an overhead of approximately 5%, cf. Fig. 6(c).

In the presence of failure some overhead is unavoidable, because communicators have to be revoked and shrinked. Our results indicate that it is negligible. We even observe that the run-time decreases for very small MTBF. At first, this seems strange. But in the presence of high failure rates large samples are likely to be aborted and only processes dealing with coarse levels are able to provide results. These processes often run on underloaded processors.

No significant overhead is measured for the "intermediate save" compared to the "late save" strategy. Since the "intermediate save" strategy performs better, regarding the range of applicability, it is advisable to use this technique.

In the plots an average over 30 FT-MLMC runs was used. Using additional runs wold increase the accuracy of the measurements.

5. Conclusion

We implemented a fault tolerant multi-level Monte Carlo (FT-MLMC) method and integrated it into the existing MLMC program ALSVID-UQ. As a fault tolerant MPI we used ULFM of which a development implementation is available, based on Open MPI. This enables our code to handle process failures.

We tested our fault tolerant version of ALSVID-UQ by solving the 2-dimensional stochastic Euler equations of gas dynamics. The FT-MLMC results of these equations have been analysed for various mean time between failures (MTBF). We showed that FT-MLMC provides an MTBF range, where fault tolerance is needed and where our method provides good results. These experiments confirm our FT-MLMC theory [2].

[6] According to George Bosilca the mentioned bug is fixed in newer ULFM versions.

Our ultimate goal is to run the FT-MLMC method on emerging massively parallel computing platforms, where failures will be unavoidable. Some parts of our findings can be transferred to the future needs. For large runs, the probability that no samples are computed, should rather shrink, as these runs will have more levels, more samples, and most samples require domain decomposition. Program crashes, due to ULFM, will surely decline, as more mature implementations will be available. With them, the runtime overhead due to fault tolerance should decrease as well. In the presence of many failures the relative error will always grow. At some failure rate the error will become unacceptably large, such that an FT-MLMC method alone will no longer perform well. However, as predicted by our theory [2], there is a considerable range, where our simple and cheap FT-MLMC procedure is beneficial.

Acknowledgment

The authors thank Jonas Sukys for the support with ALSVID-UQ, Adrian Ulrich for the support with our cluster Brutus, and the anonymous reviewers for their valuable comments and suggestions that improved the quality of the paper.

References

[1] F. Cappello. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities. *Int. J. High Perform. Comput. Appl.*, 23(3):212–226, 2009.

[2] S. Pauli, P. Arbenz, and Ch. Schwab. Intrinsic fault tolerance of Multi-level Monte Carlo methods. Tech. Report 2012-24, ETH Zürich, Seminar for Applied Mathematics, August 2012.

[3] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. Dongarra. An evaluation of user-level failure mitigation support in MPI. In J. L. Träff, S. Benkner, and J. Dongarra, editors, *Recent Advances in the Message Passing Interface*, pages 193–203. Springer, 2012. (LNCS, 7490).

[4] A. Barth, Ch. Schwab, and N. Zollinger. Multi-level Monte Carlo finite element method for elliptic PDEs with stochastic coefficients. *Numer. Math.*, 119(1):123–161, 2011.

[5] M. B. Giles. Multilevel Monte Carlo path simulation. *Oper. Res.*, 56(3):607–617, 2008.

[6] S. Mishra and Ch. Schwab. Sparse tensor multi-level Monte Carlo finite volume methods for hyperbolic conservation laws with random initial data. *Math. Comp.*, 81(280):1979–2018, 2012.

[7] S. Mishra, Ch. Schwab, and J. Šukys. Multi-level Monte Carlo finite volume methods for shallow water equations with uncertain topography in multi-dimensions. Tech. Report 2011-70, ETH Zürich, Seminar for Applied Mathematics, November 2011.

[8] *ALSVID-UQ*. Available from http://www.sam.math.ethz.ch/alsvid-uq, May 2013.

[9] J. Šukys, S. Mishra, and Ch. Schwab. Static load balancing for multi-level Monte Carlo finite volume solvers. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics (PPAM 2011)*, pages 245–254. Springer, 2012. (LNCS, 7204).

[10] A. Papoulis and S. U. Pillai. *Probability, random variables, and stochastic processes*. McGraw-Hill, Boston, 4th edition, 2002.

[11] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258. IEEE Computer Society, 2006.

[12] H. Rinne. *The Weibull Distribution: A Handbook*. CRC Press, 2009.

[13] Y. Zhao. *Parametric inference from window censored renewal process data*. PhD thesis, The Ohio State University, Columbus, Ohio, 2006.

[14] S. Mishra, Ch. Schwab, and J. Šukys. Multi-level Monte Carlo finite volume methods for nonlinear systems of conservation laws in multi-dimensions. *J. Comput. Phys.*, 231(8):3365–3388, 2012.