FETOL: A Divide-and-Conquer Based Approach for Resilient HPC Applications

Wassim Abu Abed, Kostyantyn Kucher,
and Manfred Krafczyk
Institute for Computational Modeling in Civil Engineering
Technische Universität Braunschweig
Brunswick, Germany
Emails: {abuabed, kucher, kraft}
@irmb.tu-bs.de

Markus Wittmann, Thomas Zeiser
and Gerhard Wellein
Erlangen Regional Computing Center
University of Erlangen-Nuremberg
Erlangen, Germany
Emails: {markus.wittmann, thomas.zeiser, gerhard.wellein}
@fau.de

***Abstract*—The inevitable increase of the frequency of hard and soft faults in current and future high performance computing systems motivates the need of integrated approaches to improve the resilience of such systems. In this paper, a framework for a fault tolerant environment termed FETOL implementing an approach to achieve a coordinated resilience solution is presented. FETOL is based on a software solution exploiting a Divide-and-Conquer strategy that will offer comprehensive methods on the middleware and application level to deal with various failure scenarios.**

***Keywords*—hpc; resilience; fault-tolerance; divide-and-conquer.**

I. INTRODUCTION

The increasing size and complexity of HPC (High Performance Computing) systems are two major factors that are leading to an inevitable increase of the frequency of hard and soft faults in present Peta-Flop and future Exa-Flop systems. Therefore, these HPC systems are prone to become less robust and the operating efficiency and reliability of such systems tend to deteriorate profoundly. New integrated approaches to improve the resilience of HPC systems are undoubtedly needed in order to maintain a reasonable operation of such systems. Recent literature surveys have shown that resilience cannot be efficiently realised by implementing fault tolerance mechanisms on the system level only [1]. Different application domains have different methodological requirements to achieve resilience of an HPC application. Hence, an integrated application oriented approach is mandatory.

In this paper, the concept of an application oriented framework for a fault tolerant environment termed FETOL is presented. FETOL is an abbreviation of the German translation "Fehler Toleranz" of "Fault Tolerance". FETOL is based on a software solution exploiting a Divide-and-Conquer strategy. In the next section, the background and motivation of the presented work are given. The main idea of FETOL with its central operation of breaking down the HPC application into subtasks and grouping them into individually restartable process bundles is explained in Section III. In Section IV, the system architecture and design are presented including the checkpointing and the fault tolerance mechanisms. In Section V, the proposed approach is discussed in the context of related work in the field. An evaluation of a new communication component developed as part of the proposed framework is presented in Section VI. In the last section a brief conclusion and an outlook are given.

II. BACKGROUND

Different MPI (Message Passing Interface) implementations are presently the most used parallel programming libraries in developing large-scale parallel applications. A parallel application in the context of MPI usually runs on a cluster of computing nodes in the form of different MPI processes. MPI processes are grouped together in a so-called MPI communicator that takes care of the explicit and unambiguous identification of messages and addresses respectively. Accordingly, the frequently needed communication for the purpose of exchanging data or synchronising pace among these processes is carried out via MPI library calls. In the case of a network failure or the breakdown of one of the computing nodes the state of an MPI communicator becomes undefined and the entire parallel application will come to a halt and terminate without any protective or recovery actions. This serious drawback of the current existing MPI implementations causes the interrupted parallel application to terminate irrecoverablly and thereby the entire MPI parallel application must manually be restarted from scratch or at least from the last checkpoint data.

III. THE APPROACH IN FETOL

The divide-and-conquer strategy followed in FETOL aims at implicitly overcoming the limitation in the MPI implementations that renders the parallel application irrecoverable by grouping the processes of a parallel application into more than one so-called PB (Process Bundle), see Figure 1. Each of these PBs will be executed on one or more computing nodes of the cluster. The processes within each PB communicate with each other via native MPI using a PB specific MPI communicator. An additional cross PBs communicator called BOND, see Section IV-B, will take care of the needed communication between the processes of two different PBs. This alternative communicator is based on TCP/IP and on a multi-agent architecture.

The execution of the parallel application is started by sequential or parallel initialisation of all PBs, where each PB is regarded as a separate MPI parallel application that uses a bundle specific MPI communicator. In this case, a hardware failure will only affect one of the PBs and accordingly only the respective MPI communicator will be in an undefined state. As a result, there will be no need to restart the execution of the entire parallel application and only the affected PB must be restarted. Moreover, restarting the defect PB from the very

 ISBN: 978-1-61208-310-0

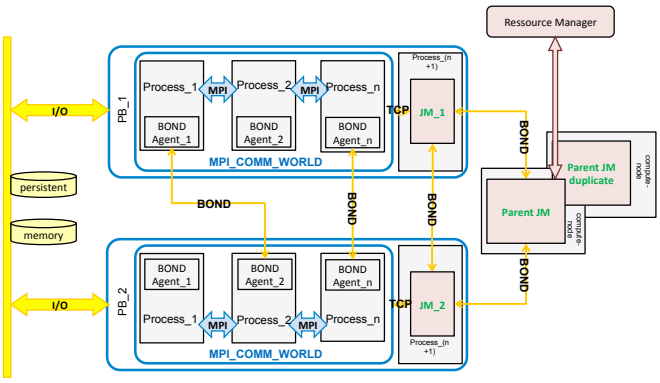

Figure 1. FETOL System architecture.

beginning can also be avoided in making use of the other PBs that are still running. The initialisation of the new PB can be done either from reconstructed data of the PBs that are still alive or from available check pointing data written by the lost PB before its crash. A coordinating middleware, called JM (Job Manager), is responsible for restoring and migrating any failed PB in cooperation with the resource manager that usually controls the computing nodes of the cluster.

IV. System Architecture and Design

Resilient high performance computing requires the collaboration between different software components. These components are: the resource manager with the associated hardware system monitoring tools, the communication libraries used to allow a parallel execution, the parallel application itself with its application level techniques of data persistency and the software monitoring and fault detection mechanisms. The collaboration between these components is managed in FETOL by introducing the JM middleware. Figure 1 schematically shows FETOL's system architecture. In the following sections the different components of this architecture will be described as well as the fault tolerance mechanism and the checkpointing approach adopted in FETOL.

A. Job Manager

The JM, as a coordinating middleware, has the task of bundling, orchestrating and extending the different functionalities of the other software components in FETOL. Different system and application information, which are delivered by the monitoring tools on the system level and the soft monitoring on the application level, will trigger the coordinated reactions of the Job Manager.

The JM should restore and migrate any failed PB. Restoring a PB includes managing and coordinating the process of retrieving application state data as well as remapping the cross PB communication channels between the restored processes and the still running processes of the other PBs. Application state data include the usual checkpointing data already stored on persistent storage or/and data sets that are reconstructed from the data of the still running processes, which are carrying computations on partitions of the application computational domain adjacent to that of the PB being restored. Migrating the restored PB to run on new hardware resources is being carried out in cooperation with the resource manager that is in control of the compute nodes of the cluster.

The system architecture shown in Figure 1 illustrates the implementation of the JM. The system JM consists of multiple software instances that run on different computing nodes. Each PB has its own JM software instance that is responsible for the specific PB. The different PB specific JM software instances are coordinated and managed by two identical central "Parent JM" software instances that operate in a synchronously parallel fashion on two different nodes of the cluster to increase the redundancy of the system. These two "Parent JM" instances assume the task of communicating with the resource manager and the responsibility of restoring any PB specific JM with failure. All software instances of the JM communicate and pass the needed information between each other using the BOND communication framework.

B. BOND

During the EU project COAST a software framework was developed to couple multiple individual software solvers in a flexible and distributed manner [2]. This framework allows distributed coupling according to agent oriented programming principles. Multi agent systems already proved their robustness and flexibility in other distributed systems [3][4].

With the acquired experiences in multi agent frameworks on HPC hardware, a successor framework coined BOND has been developed. This framework has been adapted to closely follow the programming paradigms known from MPI programmes, yet keep some of the benefits of multi agent platforms, such as robustness and scalability. The main use case of BOND is within the FETOL framework, where it serves as an alternative communication library to cover failures of MPI.

BOND is a C++ library with the corresponding API (programming interface). Bindings for Fortran and plain C are targeted for a later stage of the project. Within the C++ library, Java asynchronous socket communication is being used to deliver MPI style data buffers to a remote machine in a distributed HPC environment. BOND communication currently utilises the following primary communication calls: non-blocking send operation, blocking send operation and blocking receive operation.

On its way to the remote machine, the data is directly transferred to a user allocated target buffer, as it is common in MPI distributed HPC codes. The data buffer is never managed by the Java memory management, but always remains in native heap space. Due to this cautious data management, the Java garbage collector does not pose an overhead for data transfers, making communication speed via BOND comparable to that of MPI, see Section VI. Even high performance host channel adapters like Infiniband can be used effectively, bypassing most of the TCP/IP protocol overhead using the socket direct protocol.

C. MPI Program

The class of MPI programs considered in FETOL addresses primarily stencil based applications that are based on computational domain decomposition in the form of meshes to carry

 ISBN: 978-1-61208-310-0

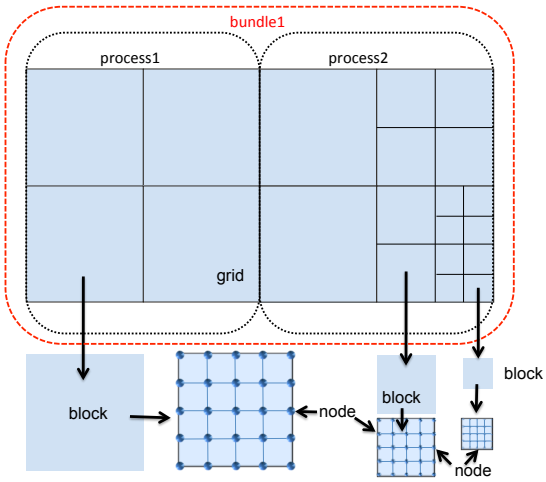

Figure 2. Data structure of the in-house LB solver and partitioning into bundles of processes.

on their calculations. LB (Lattice-Boltzmann) solvers can be regarded as an important example of this class of scientific applications. Our in-house Lattice-Boltzmann flow solver VirtualFluids [5] is an adaptive MPI based parallel application which is comprised of various cores. The software framework is based on object-oriented technology and uses tree-like data structures. The flow region is divided into discrete blocks on the basis of an Octa-tree. These discrete blocks contain the numerical sub-grids of the computational domain in the form of equally dimensioned matrices of nodes per block, see Figure 2. This structure has many advantages. The use of uniform girds in form of matrices within blocks allows the use of efficient algorithms and requires less computing resources since direct addressing is possible and the cache memory of CPU can be better utilised. Restricting the communication on the block-edges/-areas reduces the complexity of parallelisation. These data structures are also suitable for hierarchical parallelisation using a combination of PThreads and MPI and dynamic load balancing.

D. Check-Pointing Approach

Data persistency approaches in the field of fault-tolerant HPC applications can be divided into Message-logging and Check-pointing approaches. For the class of the scientific applications considered in FETOL, the approach of message-logging is not practical. Usually scientific applications of this class exchange too much data. In the case of recovery the live processes would have to wait too long until the recovered process computes the current actual consistent state and thus, the overall efficiency of the resilience mechanism would deteriorate unacceptably.

In FETOL, an application-level checkpointing approach is adopted and implemented in the framework of the in-house LB solver. In contrast to the widely used system-level checkpointing, where the system stores the complete state of the application independently, different mechanisms of data reduction can be implemented minimising the volume of stored data per checkpoint.

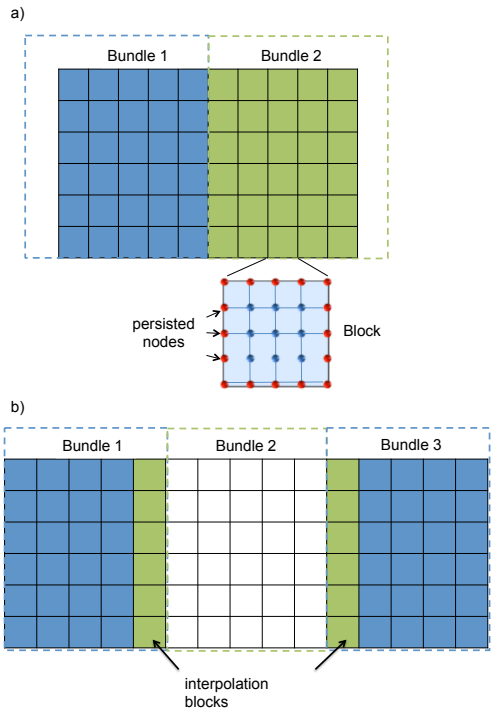

Figure 3. Data persistency (a) the boundary nodes of all blocks are stored. (b) recovery depending on neighbouring bundles

The individual steps of the periodical checkpointing procedure in VirtualFluids and a mechanism to deal with data consistency are given in the following: (1) The application periodically serialises and stores the state data of the corresponding blocks of the grid of each MPI process in the file system. (2) After saving each process state data the process internally sets a checkpoint counter variable. (3) This action is then synchronised by a root process that sets a global environment variable, that indicates the last successful checkpointing, at which the state of the application is consistent.

Two different mechanisms of data recovery after failure are implemented in VirtualFluids. The first one is based on a data reduction strategy that minimises the volume of the stored data per checkpoint. When checkpointing the simulation data of a process, only the information in the boundary nodes of each block is saved, see Figure 3 (a). The rest of the omitted information are then interpolated for the internal nodes of each block during the data recovery procedure. The second recovery mechanism is applied when no checkpointing data for a whole bundle can be retrieved at all. The missing data are then approximated from the data of the still running processes, which are carrying computations on partitions of the application computational domain adjacent to that of the bundle being restored, see Figure 3 (b).

E. Fault Tolerance Mechanism

After specifying the needed resources and the executable binary of the Parallel Application by the user the execution of the JM is started on the Cluster. The Resource Manager assigns the needed resources, which were specified by the User, plus additional so-called recovery resources to the JM. The JM

 ISBN: 978-1-61208-310-0

starts the execution of the Parallel Application on the Cluster. The JM listens permanently to and analyses the information delivered by the Hardware and Software Monitors. The JM terminates when the Parallel Application successfully ends.

The occurrence of hardware faults will prevent the successful execution of the Parallel Application, which will result in two different scenarios. The first scenario happens when the JM identifies - according to the analysis of the information received from the Hardware and Software Monitors - any running vulnerable process that might put the execution of the Parallel Application at risk. In this case the JM initiates an active fault-tolerance action by stopping and then restoring the execution of the process bundle containing the vulnerable process on new resources and the execution of the Parallel Application continues.

The second scenario can occur, if the Hardware and/or Software Monitors are totally absent as software components or fail to report a failure. In this case the Parallel Application might fail to continue its execution unnoticed by the JM. For this reason, the BOND framework, which is imbedded in the Parallel Application, will support a passive fault-tolerance action of the JM and takes the responsibility of reporting any failed process bundle to the JM. The JM then restores the failed process bundle and the execution of the Parallel Application continues.

The step of restoring a process bundle requires the retrieval of the process state data for each process of the process bundle being restored from the Storage System. Here the Parallel Application retrieves the process state data stored in form of checkpointing data from the Storage System or approximates a set of process state data, using the still running processes' data, i.e, the corresponding "neighbouring" PBs.

The JM then remaps the communication channels between the processes being restored and those still running. At the end the JM migrates and starts the restored process bundle on new nodes of the Cluster using its extra recovery resources. If the extra recovery resources of the JM are exhausted, the JM demands new recovery resources from the Resource Manager. The latter assigns then the demanded resources and the migration of the process bundle can be successfully accomplished.

V. Related Work

Different approaches that address the problem mentioned in Section II have recently been proposed and implemented. These approaches can be grouped into the two categories: MPI based and non MPI based approaches.

MPI based approaches [6][7][8]: In [6] a ULFM (User Level Failure Mitigation) approach is presented, where a set of five new interfaces are added to the MPI implementation. In contrast to [6], in this paper, no extension of the MPI standard or implementation is suggested. Hence, protective actions are located entirely outside the MPI implementation and the use of arbitrary MPI implementations is possible. The recovery processes is driven by three components (the Job Manager, the additional communication framework BOND and the parallel application). Only the MPI communicator of a group of processes Bundle is replaced by restarting a new mpirun instance that spawns replacement processes within a new MPI communicator. The MPI communication performance is essentially maintained inside each bundle.

The approach in [7] aims at enabling MPI implementations to support Algorithm-based Fault Tolerant techniques, see also [11]. It avoids any periodic checkpointing by storing the application state only after a failure is detected. The MPI runtime is augmented with a failure detection service and the MPI implementation is modified. In FETOL, instead of relaunching the entire MPI application, only the Process Bundle, which is a subset of the parallel application, is relaunched. The Checkpoint-on-failure approach in [7] can be integrated bundle-wise in FETOL to support an ABFT (Algorithm-Based Fault Tolerant) like technique used to recover data from checkpoints.

The fault tolerance mechanism of the Job Pause Service in [8] is implemented within LAM/MPI using the BLCR [9] as a checkpointing library. The fault tolerance mechanism allows live processes to remain active after a notification of a process failure. The live processes will roll back to the last checkpoint and retain the internal communication links. Failed processes are dynamically replaced by new ones on spare nodes before resuming from the last checkpoint.

The fault tolerance mechanism in [8] resembles the one proposed in this paper to a fair extent. For example, the Job Manager can be compared to the Scheduler daemon. BOND can be compared to the scalable communication infrastructure used to notify the active nodes about the replacement nodes and reconfigure the communication infrastructure. However, the following differences are significant: the fault tolerance mechanism in FETOL has the scope of bundles of processes as opposed to a single process; No modification to the MPI implementation is necessary; Checkpointing is carried out on the application level avoiding the lack of flexibility of the system level checkpointing.

Non MPI based approaches [10]: In [10] an object-oriented parallel programming library for C++ called Charm++ is presented. It differs from traditional message passing programming libraries (such as MPI) in that Charm++ is message-driven. Furthermore, it provides a methodology and a virtualisation infrastructure in which the programmer decomposes the data and computation in the program without worrying about the number of physical processors on which the program will run. The runtime system is in charge of distributing those objects among the processors.

FETOL does not offer the sophistication of Charm++ [10] and keeps the fault tolerance mechanism as simple as possible reducing the complexity of the implementation and the usage. Therefore, the user application can still rely on its favourite MPI implementation. In addition, the asynchronous communication framework BOND provides an automatic overlap of communication with computation, which facilitates the fault-tolerance mechanism in FETOL.

In the following, two categories of data recovery approaches that are tangential to the approach followed in this paper are also identified and briefly summarised.

Algorithm-based fault-tolerance [11][12]: The most significant and relevant aspect of ABFT techniques [11] is that

 ISBN: 978-1-61208-310-0

failures can be tolerated without checkpointing or message logging. Per definition as found in the literature, for example in [7], an ABFT uses mathematical and algorithmic properties to reconstruct failure-damaged data and to complete operations despite failures. The Algorithm-Based Checkpoint-Free Fault Tolerance Technique in [12] extends the ABFT idea to recover applications from failures, in which the failed process stops working and the data are totally lost.

In FETOL, a data recovery strategy similar to ABFT is adopted in that mathematical interpolations are used in two ways: first, in recovering reduced checkpointing data, second, in recovering and restoring missing data from living processes. Moreover, the assumptions about the capabilities of the runtime environment mentioned in [12] are guaranteed by FETOL.

Checkpointing and message-logging [13][14]: The partial message logging protocol proposed in [13] is based on process clustering and a hierarchical rollback-recovery protocol that applies different protocols for the communications inside a cluster of processes and for the communications among the cluster. The group-based Checkpoint/Restart solution in [14] combines coordinated checkpointing and message logging. In FETOL, message logging is totally avoided, since the properties of the scientific parallel application considered in FETOL make the use of message logging disadvantageous in terms of storage size and time overhead of recovery. One more distinction between FETOL and [14] is that the parallel application is responsible for receiving checkpoint requests and writing and reading checkpoints, instead of modifying mpirun. One similarity to the hierarchal rollback-recovery protocols mentioned in [13] that should be stated is the division of processes in groups called Bundles and the application of different communication protocols inside and among the different bundles. Therefore, FETOL can be regarded as a hierarchical rollback-recovery protocol.

VI. Evaluation

The low-level ping-pong benchmark was used to evaluate the bandwidth performance of BOND in comparison to Intel MPI, for given different message sizes, see Figure 4. The different tests were conducted on a cluster based on two-socket compute nodes equipped with 16-core AMD Opteron 6134 Magny Cours processors. The cluster is also equipped with a GE (Gigabit Ethernet), which can achieve around 125 MB/s, as well as with a fully non-blocking IB (QDR-InfiniBand) network with a bandwidth of ca. 3 GB/s. An implementation of the ping-pong benchmark using BOND was compared to the one included in the Intel MPI benchmarks [15] compiled with Intel MPI. The different tests have shown the following results, see Figure 4. Over GE both implementations can sustain nearly the full bandwidth. Moreover, both implementations show the same quantitative bandwidth when using IPoIB (IP protocol over the IB interconnect), however Intel MPI achieves a slightly higher bandwidth for messages smaller than 64 kB and BOND for messages larger than that. The usage of SDP (the Sockets Direct Protocol), which provides the advantages of RDMA transfers from IB to IP connections, only brings a benefit for Intel MPI. BOND stays nearly on the same bandwidth level in a similar way as in the IPoIB test version. Interestingly, the bandwidth for BOND drops below 1 MB/s when message sizes fall in the interval between 16 kB and

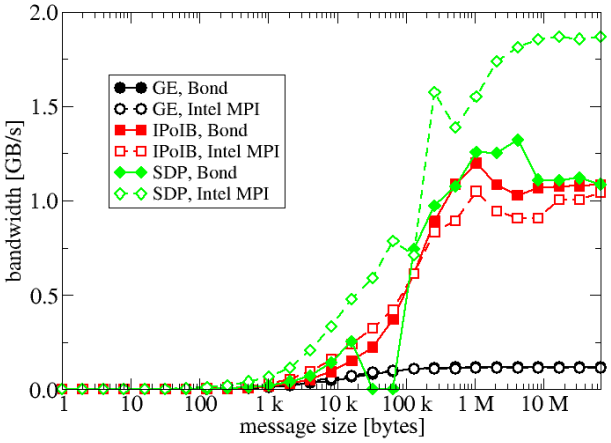

Figure 4. Performance evaluation - BOND in comparison to Intel MPI

128 kB. It is suspected that the unchanged performance of BOND with SDP and the bandwidth drop could be due to implementation details of the Java SDP interface.

VII. Conclusion

The full implementation of the divide-and-conquer strategy for a coordinated resilience of an HPC application presented in this paper is still under development. A ”proof-of-concept” implementation is presently developed and first benchmarking tests of the new communication component BOND were presented in this paper. The presented results show that there are no substantial side effects of using BOND beside MPI. On the contrary, BOND might even outperform MPI in some cases. The implementation and test of a procedure for passive fault-tolerance is being addressed at the time of this writing. For the passive fault tolerance the BOND framework is exploited as a pseudo-monitoring tool that captures and reports any error collectively and indifferently as a failure of a process bundle. Addressing different categories of faults will then be handled in the context of implementing an active fault-tolerance approach that utilises a hardware monitoring tool on the system level and some software monitoring mechanism on the parallel application level. A fundamental problem that still has to be addressed for implementing an efficient active fault-tolerance is the definition and test of quality metrics that quantify and indicate the vulnerability of a process bundle. The identification of system parameter thresholds that indicate when the vulnerability of a PB is critical is a trade-off between the increased risk of complete failure, in case no intervention was undertaken, and the overhead incurred by a fault prevention measure, in which the execution of a still running PB, but yet considered as vulnerable, is stopped and restarted as a precaution.

Acknowledgement

The authors would like to thank the German Federal Ministry of Education and Research - BMBF for supporting this work (Grant Number 01 IH11011).

 ISBN: 978-1-61208-310-0

REFERENCES

[1] J. Daly (eds.), Inter-Agency Wrokshop on HPC Reilience at Extreme Scale, National Security Agency Advanced Computing Systems, February 2012.

[2] Institute for Computational Modeling in Civil Engineering - Technische Universität Braunschweig, https://www.irmb.bau.tu-bs.de/muscle/, retrieved: Sep. 10. 2013.

[3] E. Cortese, F. Quarta and G. Vitaglione, "Scalability and Performance of JADE Message Transport System," AAMAS Workshop, Bologna, 2002.

[4] A. Helsinger, M. Thome and T. Wright, "Cougaar: a scalable, distributed multi-agent architecture," Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 2004, 2, IEEE, 2004, pp. 1910–1917.

[5] M. Schönherr, K. Kucher, M. Geier, M. Stiebler, S. Freudiger and M. Krafczyk, "Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs," Int. J. Comp. Math. App. 61, 2011, pp. 3730–3743.

[6] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca and J. Dongarra, "An Evaluation of User-Level Failure Mitigation Support in MPI," Proceedings of Recent Advances in Message Passing Interface 19th European MPI Users Group Meeting, EuroMPI 2012. Springer, Vienna, Austria, Sep. 2012, pp. 193–203.

[7] W. Bland, P. Du, A. Bouteiller, T. Herault, G. Bosilca and J. Dongarra, "A Checkpoint-on-Failure Protocol for Algorithm-Based Recovery in Standard MPI,"Kaklamanis et al. (eds.) Euro-Par 2012, LNCS, vol. 7484, Springer-Verlag, Berlin Heidelberg, 2012, pp. 477–488.

[8] C. Wang, F. Mueller, C. Engelmann and S. L. Scott, "A Job Pause Service under LAM/MPI+BLCR for Transparent Fault Tolerance," International Parallel and Distributed Processing Symposium, IPDPS 2007, IEEE International 2007, pp. 26–30.

[9] Berkeley Lab, http://crd.lbl.gov/groups-depts/ftg/projects/current-projects/BLCR, retrieved: Sep. 13. 2013.

[10] Parallel Programming Laboratory - University of Illinois at Urbana-Champaign, http://charm.cs.uiuc.edu/research/charm/, retrieved: Sep. 10. 2013.

[11] K. Huang and J.A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," IEEE Transactions on Computers, Vol. c-33, No. 6, 1984, pp.518–528.

[12] Z. Chen and J. Dongarra, "Algorithm-Based Fault Tolerance for Fail-Stop Failures," IEEE Transactions on Parallel And Distributed Systems, Vol. 19, No. 12, 2008, pp. 1628–1641.

[13] T. Ropars, A. Guermouche, B. Uar, E. Meneses, L.V. Kal and F. Cappello, "On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications," In Jeannot, E. , Namyst, R., Roman J. (eds.) Euro-Par 2011, LNCS 6852, Part I, Springer, Heidelberg 2011, pp. 567–578.

[14] J. C. Y. Ho, C. Wang and F. C. M. Lau, "Scalable Group-based Checkpoint/Restart for Large-Scale Message-passing Systems." IEEE International Symposium on Parallel and Distributed Processing, IEEE, 2008, pp.1–12.

[15] Intel Corp, Intel MPI benchmarks, http://software.intel.com/en-us/articles/intel-mpi-benchmarks, retrieved: Sep. 12. 2013.

 ISBN: 978-1-61208-310-0