Performance Measurement and Analysis of Transactional Memory and Speculative Execution on IBM Blue Gene/Q

Jie Jiang[1,2⋆], Peter Philippen[1], Michael Knobloch[1], and Bernd Mohr[1]

[1] Forschungszentrum Jülich GmbH,
Institute for Advanced Simulation,
Jülich Supercomputing Centre,
52425 Jülich, Germany
{j.jiang,p.philippen,m.knobloch,b.mohr}@fz-juelich.de

[2] National University of Defense Technology,
School of Computer Science,
Changsha, Hunan Province, 410073, China
jiangjie@nudt.edu.cn

Abstract. The core count of modern processors is steadily increasing, forcing programmers to use more concurrent threads or tasks to effectively use the available hardware. This in turn makes it increasingly challenging to achieve correct and efficient thread synchronization. To support the programmer in this task, IBM introduced hardware transactional memory (TM) and speculative execution (SE) in their Blue Gene/Q system with its 16-core processor, which permits to run 64 simultaneous hardware threads in SMT mode. TM and SE allow for parallelization when race conditions may happen, however upon their detection the respective parts of the execution are rolled back and re-executed serially. This incurs some overhead and therefore usage must be well justified. In this paper, we describe extensions to the community instrumentation and measurement infrastructure Score-P, allowing developers to instrument, measure, and analyze applications. To our knowledge, this is the first integrated performance tool framework allowing to analyze TM/SE programs. We demonstrate its usefulness and effectiveness by describing experiments with benchmarks and a real-world application.

Keywords: Parallel Programming, Performance Analysis, Transactional Memory, Speculative Execution, Blue Gene/Q

1 Introduction

The number of cores available in modern processors as well as the number of processors inside cache-coherent shared-memory nodes is steadily increasing, es-

⋆ This work is partially supported by the National Basic Research 973 Program of China under Grant No.61312701001, the National High Technology Research and Development Program of China under Grant No.2012AA01A309.

pecially in high-end servers and HPC cluster systems. This forces parallel program developers to use more concurrent threads or tasks to effectively use the available hardware, in turn making it increasingly challenging to achieve correct and efficient thread synchronization.

IBM's latest HPC architecture, the Blue Gene/Q, is based on a 16-core PowerPC A2 processor, running up to 64 simultaneous hardware threads in symmetric multi-threading (SMT) mode [1]. To alleviate the implementation of correct and efficient thread synchronization, IBM introduced hardware transactional memory (TM) and speculative execution (SE). The interface to the TM and SE hardware features of the Blue Gene/Q memory subsystem is based on C/C++ pragmas and Fortran directives[3] similar to the ones in the OpenMP specification. The TM programming model is based on an abstraction called a transaction. It is a single-entry and single-exit code block enclosed by a "`tm_atomic`" directive. It can be used for atomic or critical regions in the code where data access race conditions are expected to be rare and thus the locking overhead in the race-free instances of the region can be avoided. For SE, the corresponding directive has similar semantics as an OpenMP loop work-sharing construct. For example, the "`speculative for`" directive mimics an "`omp parallel for`" directive with the additional guarantee to maintain sequential semantics of the code, i.e., the result corresponds to the result of an execution by a single thread. So, TM and SE both allow for parallelization even when race conditions may happen, however upon their detection the respective parts of the execution are rolled back and re-executed serially. However, the benefit of the parallel execution must outweigh the extra management overhead. To help application developers to evaluate the effectiveness of using TM and SE constructs in their codes, the IBM compiler runtime provides a TM/SE monitoring API which allows to collect executions statistics for TM and SE constructs.

In this paper, we describe extensions to the parallel program performance analysis framework Score-P [2], which allows developers to instrument, measure, and analyze MPI, OpenMP, or hybrid MPI/OpenMP parallel applications which also use TM and SE constructs. This integration allows the user to analyze all aspects of parallel performance in one tool environment and to study dependencies and relationships between parallel constructs from the different programming paradigms. For the instrumentation of directive-based parallel programming paradigms, Score-P uses the Open Pragma And Region Instrumenter (OPARI2) tool, which was enhanced to handle IBM TM and SE directives. Measurement results are stored as summary profiles which can be analyzed and viewed by the Cube [3] performance report viewer.

The main contributions of the work described in this paper are:

- A generic extensible tool for automatic instrumentation of directive-based parallel programming paradigms including OpenMP and IBM TM/SE.
- An integrated performance tool framework allowing to analyze MPI, OpenMP, or hybrid MPI/OpenMP parallel programs using TM and SE constructs. To our knowledge, this is the only tool set providing this capability.

[3] As in the OpenMP specification, in this paper we will use the term directive for both

The rest of the paper is organized as follows: Section 2 gives a brief overview on related work. Section 3 introduces the performance tool components which were used, adapted, and enhanced, including the IBM TM/SE monitoring API, OPARI, Score-P and Cube. The experiments to evaluate the usefulness and effectiveness of the introduced extensions to our tool infrastructure are described in Section 4. Finally, conclusions and a description of future work close the paper.

2 Related Work

Research on Transactional Memory (TM) has a long history, being first introduced by Herlihy and Moss [4] in 1993 as a theoretical extension to microprocessors. Subsequent research shifted towards Software Transactional Memory (STM) [5], i.e. software ensures the atomicity of the transactions and organizes the rollbacks. Today, STM implementations are available for many programming languages, either as language feature (e.g. Closure) or as a library (e.g. for C/C++, C#, Java).

Several research groups proposed analysis techniques for software transactional memory using different methods. Ansari et al. [6] extended an STM framework to obtain profiling data while Zyulkyarov et al. [7] track data structures that conflict in transactions and determine their influence on the performance of the application. Tracing of transactional memory applications was introduced by Lourencco et al. [8], using a similar approach like the group of Ansari. However, due to the relatively high overhead of STM, this approach is of minor relevance to real-world applications in the field of high-performance computing [9].

IBM presented the first commercially available hardware transactional memory (HTM) system in the Blue Gene/Q (BG/Q) supercomputer [1]. Wang et al. [10] and Schindewolf et al. [11] evaluated the HTM implementation on BG/Q using various benchmarks to determine which applications may benefit from TM. Scientific application developers begin to embrace HTM; performance studies have been performed by Kunaseth et al. [12] for molecular dynamics applications and by Schindewolf et al. [13] for the conjugate gradients method.

On the other hand, the Speculative Execution (SE) functionality of BG/Q has not yet been so well investigated. To the best of our knowledge, no extensive performance study for SE has been performed.

Bihari et al. [14] made a case for adding directives for transactional memory to the OpenMP specification. The importance of a standard way to express TM constructs became visible with the work of Yoo et al. [15], who evaluated the performance of the recently introduced Transactional Synchronization Extensions of Intel's Core architecture processors.

3 Tool Implementation

To gain insight into the behavior and especially into the impact on performance of the new transactional memory and speculative execution features on IBM's

TM record	SE record
`typedef struct TmReport_s {`	`typedef struct SeReport_s {`
`unsigned long hwThreadId;`	`unsigned long totalNONSpecCommitted;`
`unsigned long totalTransactions;`	`unsigned long totalSpecCommitted;`
`unsigned long totalRollbacks;`	`unsigned long totalRollbacks;`
`unsigned long totalSerializedJMV;`	`unsigned long totalSerializedJMV;`
`unsigned long totalSerializedMAXRB;`	`unsigned long totalSerializedMAXRB;`
`unsigned long totalSerializedOTHER;`	`unsigned long totalSerializedOTHER;`
`} TmReport_t;`	`} SeReport_t;`

Table 1. Structure used by reporting functions for TM counters

BlueGene/Q architecture, we added support into OPARI2 and to the performance measurement framework Score-P. Source-to-source translation is used to insert probe functions into the application code to instrument the regions of the code that make use of TM and SE. These probe functions are implemented in one of the measurement libraries of Score-P, the so-called TM/SE adapter, and process the data provided by IBM's TM/SE monitoring API to make it usable by the measurement system. The data is recorded and stored in profiles which can be examined with Cube.

This section introduces the IBM TM/SE monitoring API and presents the extensions to OPARI2 that were necessary to perform the instrumentation of the TM/SE directives. Next, the measurement system Score-P and the newly implemented adapter for TM/SE are briefly described. Finally, this section concludes with a detailed description of the newly developed analysis possibilities for transactional memory and speculative execution.

3.1 IBM TM/SE Monitoring API

The SMPRT runtime system on IBM BlueGene/Q provides several intrinsics for application programmers and tool developers to collect accumulative statistic for TM/SE regions.

tm_get_stats(TmReport_t * stats) collects the relevant accumulative statistics for all TM regions that a particular hardware thread has executed up to the point of the call, and stores it in a record of type *TmReport_t* as shown in Table 1, left. The main fields of this record include the hardware thread ID, the total number of transactions, the total number of rollbacks, and the total number of serialized executions (instead of successful speculative executions), caused either by JMV[4] conflicts, the maximum number of rollbacks reached, or other reasons. This function can be called both at the beginning and the end of a transaction, the difference reflecting the contribution of the enclosed region. To get thread-specific values, it should be used inside parallel regions (like OpenMP parallel regions).

[4] Jail Mode Violations occur in case of irrevocable actions, e.g. I/O.

tm_get_all_stats(TmReport_t * stats) behaves similarly, but it provides the accumulative statistics of all the TM regions that all hardware threads have executed up to the point of the call. This function should be used outside of parallel regions.

se_get_all_stats(SeReport_t * stats) updates the provided record (see Table 1, right) with the sum of the statistics of all the SE regions that all hardware threads have executed up to the call. The statistic counters for speculative execution include the total number of chunks committed by none speculative threads, the total number of chunks committed by speculative threads, the total number of rollbacks for speculative threads, the total numbers of serializations (caused by JMV conflicts, due to reaching the maximum number of rollbacks, and due to other reasons like buffer overflows, hardware races, etc.).

3.2 Instrumenting TM/SE Programs

We use the TM/SE monitoring API described above for collecting runtime accumulative statistics about the execution of TM/SE regions in the application. The necessary instrumentation can be done in various ways; we use source-code based instrumentation to be able to attribute performance data to user-level constructs easily and in a portable way.

The Open Pragma And Region Instrumenter (OPARI2) is a source-to-source instrumentation tool that inserts probe functions and code segments into an application's source code. OPARI2 is developed based on OPARI from the Scalasca performance analysis tool set [16]. The original version was designed to detect and instrument OpenMP directives in C/C++ and Fortran programs. It reads the source file line by line, detects OpenMP directives and runtime functions ignoring strings and comments, and instruments OpenMP constructs by inserting functions as defined by the POMP2 interface [17].

All directives that are to be instrumented are stored in an internal table. While parsing the source code, OPARI2 checks the table whenever a directive is detected. If the directive is to be instrumented, this is done at the beginning and at the end of the source-code region associated with the directive.

Support for TM/SE program instrumentation was integrated into OPARI2 under the precondition of enhancing OPARI2 to have a more modular architecture. The goal was to support different directive-based parallel programming paradigms, starting with OpenMP and IBM's TM/SE, but also keeping OpenACC and Intel MIC LEO (language extensions for offload) in mind. OPARI2 now maintains an internal table of all supported paradigms and directives. Each entry of the table includes the paradigm type, directive name, a flag indicating whether this specific directive should be instrumented, as well as two pointers to functions which perform the necessary instrumentation at the beginning and at the end of the associated source-code region. These directive-specific definitions form the basis of the modularized OPARI2, which makes it straightforward to support new paradigms and directives in the future.

Original code	Instrumented code

`#pragma speculative for`
`{`
`  ...`
`}`
	`PTLS_Speculativefor_enter( int* id,`
	`  const char context_info[] );`
	`#pragma speculative for`
	`{`
	`  ...`
	`}`
	`PTLS_Speculativefor_exit( int* id );`

`#pragma speculative sections`
`{`
`#pragma speculative section`
`  {`
`  ...`
`  }`
`  ...`
`}`
	`PTLS_Speculativesections_enter(`
	`  int* id, const char context_info[] );`
	`#pragma speculative sections`
	`{`
	`#pragma speculative section`
	`  {`
	`  PTLS_Speculativesection_begin(`
	`  int* id, const char context_info[] );`
	`  ...`
	`  PTLS_Speculativesection_end( int* id );`
	`  }`
	`  ...`
	`PTLS_Speculativesections_exit( int* id );`
	`}`

`#pragma tm_atomic`
`{`
`  ...`
`}`
	`PTLS_Tm_atomic_enter( int* id,`
	`  const char context_info[] );`
	`#pragma tm_atomic`
	`{`
	`  ...`
	`}`
	`PTLS_Tm_atomic_exit( int* id );`

Table 2. Exemplary instrumentation of TM and SE directives for C/C++

The instrumentation of IBM's transactional memory and speculative execution directives was enabled by defining and adding definitions for all TM/SE directives. That is, new table entries have been created for the `tm_atomic`, `speculative for`, `speculative do`, `speculative sections` and `speculative section` directives. The instrumentation is carried out according to the transformation rules as shown in Table 2.

3.3 Measuring TM and SE Programs

To actually measure programs employing the TM/SE techniques the instrumented executable needs to be linked to a measurement library which implements the inserted probe functions. Therefore, we extended the Score-P measurement framework accordingly.

The Score-P Instrumentation and Measurement Infrastructure is a community-driven software framework for recording profiles and traces of paral-

lel program execution [2]. The application under investigation is automatically instrumented, by means of a number of different techniques, and linked to a set of libraries that implement the respective probe functions. Each invocation of a probe function is translated into measurement events such as enter/exit of code regions, or acquire/release of locks. Different metrics like number of visits, time spent in a region, bytes transferred over a network are associated with these events. Furthermore hardware counters providing information about cache misses or floating point operations can be recorded.

There are two main modes of recording and storing data in Score-P: profiling and tracing. In a profile, summarized data is recorded for each callpath executed by the program. Times and number of visits are aggregated; minimum, maximum and average values are stored. The values of performance counters are also recorded. In contrast, in a trace every single instance of an event is recorded. This yields a very detailed view of the program run but comes at the cost of high memory demands during measurement and for storing the trace file itself.

Different methods for performing the instrumentation of an application are available. Many compilers allow for automatic instrumentation of user functions. Here, the compiler inserts probe functions at entries and exits of functions and supplies source-code information. To instrument directive-based parallel programming paradigms, we use OPARI2 as described in Section 3.2. To record MPI-specific events and metrics, PMPI interposition wrappers are used. For analyzing programs that run on GPUs, the CUDA Profiling and Tools Interface (CUPTI) is supported as well.

Each of the aforementioned instrumentation techniques inserts different types of probe functions which provide different types of information to the actual measurement system. To provide the measurement core that records profiles or traces with consistent data, Score-P contains a number of adapters, each taking care of implementing the probe or wrapper functions for a specific kind of instrumentation.

A TM/SE adapter was added in Score-P to enable the measurement of code regions making use of the transactional memory and speculative execution functionality provided by the IBM compilers. These regions, which are instrumented with OPARI2, are first registered with the measurement system. During registration, the type of TM/SE directive is stored together with source code information, consisting of file name and line numbers. Furthermore, the measurement system provides a unique numerical id, which is passed as parameter to the probe function calls surrounding the corresponding TM/SE regions (see Table 2). This allows quick access to the respective region information.

When a region is entered, interface functions provided by the TM/SE runtime are used to obtain data about the number of transactions and rollbacks as well as information about how much of the execution needed to be serialized due to JMV conflicts, too many rollbacks, and other causes, such as buffer overflows, race conditions and concurrent TM/SE regions. These values are passed as custom metrics to the measurement system. The measurement core takes care of keeping count of the number of visits to each region as well as the time spent inside.

4 Experimental Evaluation

In this section, we evaluate our approach with two examples. The first is a quasi-random field update kernel that occurs in similar form in many scientific applications. The second is MP2C, a molecular dynamics application that scales up to the whole JUQUEEN, a 28-rack Blue Gene/Q system at the Jülich Supercomputing Centre.

4.1 Update Kernel

A kernel found in many scientific applications, especially in the area of plasma physics, is an update of charge and power densities of large arrays of particles, in total 6 entries per volume cell. First the values are interpolated and then a reduction on the arrays is performed. Here, multiple threads may concurrently access the same location.

```
!$OMP PARALLEL DO private(xa, i, j1, j2, f1, f2, ci)
do i=1,5000000
  xa = x(i)*oodx
  j1 = aint(xa)
  j2 = j1+1
  f2 = xa-j1
  f1 = 1.0-f2
  ci = charge(i)
  !TM$ TM_ATOMIC SAFE_MODE
  rho(j1,ci) = rho(j1,ci) + re*f1
  rho(j2,ci) = rho(j2,ci) + re*f2
  !TM$ END TM_ATOMIC
end do
!$OMP END PARALLEL DO
```

Listing 1. Update Kernel – TM version

Listing 1 shows an example of such a kernel, although in a very simplified form. In each iteration, it performs a quasi-random update of two entries of an array of about 19 MB, which gives a conflict probability of ∼8e-7, so it seems a good candidate for TM.

Figure 1 shows a Cube screenshot of an execution of this kernel on one node of BG/Q with one process and eight threads. Cube's main window consists of three coupled tree-browsers. These show, from left to right, the metric tree, call tree and system tree. A selection of an item in one tree shows the distribution of the value associated with this item in the tree(s) to the right. In this example, the total number of transactions is selected in the metric tree, and we see the expected five million transactions. The call tree in the middle pane shows that

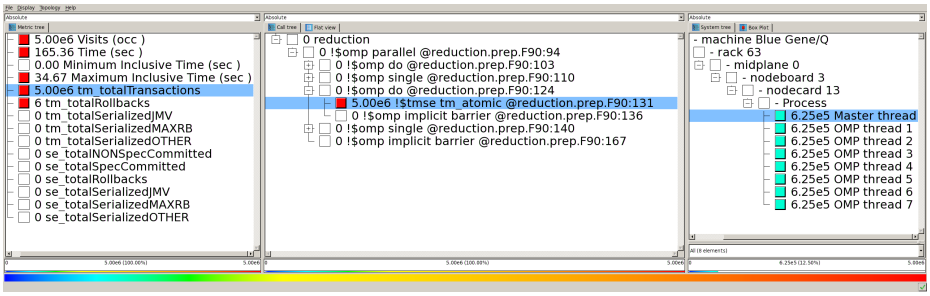

Fig. 1. Cube screenshot of the TM implementation of the update kernel showing five million total transactions distributed homogeneously among the threads.

they all originate from one TM region. The right pane, the system tree, shows that each thread completed 625,000 transactions.

While this seems to be a perfect kernel for TM, with hardly any rollbacks (6 in this example), it has to be noted that the uninstrumented TM implementation is 2 times slower than an implementation with OpenMP atomics and 3 times slower than an implementation with OpenMP reduction. This can also be easily investigated with our toolset. This shows that the tool gives correct information, but a baseline comparison to evaluate TM/SE benefits is still necessary. A detailed performance analysis of this kernel can be found in [18], where tuning opportunities are also shown which are not reflected in our measurements.

4.2 MP2C

MP2C [19] - Massively Parallel Multi-Particle Collision Dynamics - implements a hybrid representation of solvated particles in a fluid. Solutes are simulated atomistically by classical molecular dynamics (MD) which is coupled to the solvent, described by the Multi-Particle-Collision-Dynamics method (MPC). In this work we focus on the MPC part, which can be used as stand-alone implementation for particle-based hydrodynamics. The application is written in Fortran 90 and parallelized with MPI and OpenMP, which are used throughout the code. We investigated the cell collision kernel containing an OpenMP loop counting the particles in a cell and updating a list. In the initial version, this update is guarded with an OpenMP critical directive. We investigated alternative implementations with both TM and SE for this critical section. In the TM case, the OpenMP critical was replaced by an TM atomic, in the SE case the whole loop was executed speculatively.

Figure 2 shows a Cube screenshot of the TM version of the code. We see that 800 million transactions were issued and 560 million rollbacks occurred, i.e. a rollback ratio of 70%. And even worse, more than 3.5 million iterations were serialized because the maximum number of rollbacks was reached. So TM is not a good choice to replace the OpenMP critical in this case.

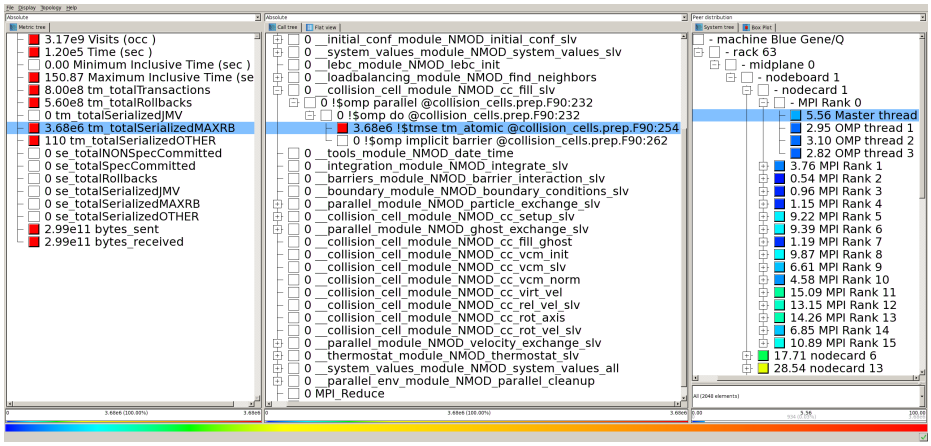

Fig. 2. Cube screenshot of the TM implementation of MP2C. It shows a high variation of serializations due to max. rollbacks among the threads.

A much better result was achieved with SE as shown in Figure 3. Here the rollback ratio is only 7% and no serializations occurred. In the system pane this screenshot shows a boxplot of the distribution of rollbacks among the processes, with a lower quartile of 3110, an upper quartile of 4410 and a median of 3660, which seems a reasonable distribution.

This example shows that our enhanced tool set easily allows to investigate TM/SE-related performance issues in parallel applications. Furthermore it also allows to compare these results with other implementations like plain OpenMP within the same environment.

5 Conclusion and Future Work

In this paper we presented a unique integrated performance tools framework to measure and analyze applications using IBM TM/SE directives. To this end, we modularized the OPARI2 source-to-source instrumenter to be easily extendable to directive-based programming paradigms other than OpenMP. A respective adapter was added to the measurement infrastructure Score-P. This adapter uses the existing TM/SE monitoring API to query information about the execution of single TM/SE regions. The resulting profile reports can be visualized and analyzed with the Cube performance report viewer. With two examples we proved the applicability of the tool and showed the added value to performance analysis of parallel applications.

One disadvantage of our approach is that TM instrumentation may add significant overhead, especially for regions with small workload, as `tm_get_stats()` gets called twice per region. We will investigate methods to reduce that overhead, e.g. by minimizing the number of calls to the IBM monitoring API. However,

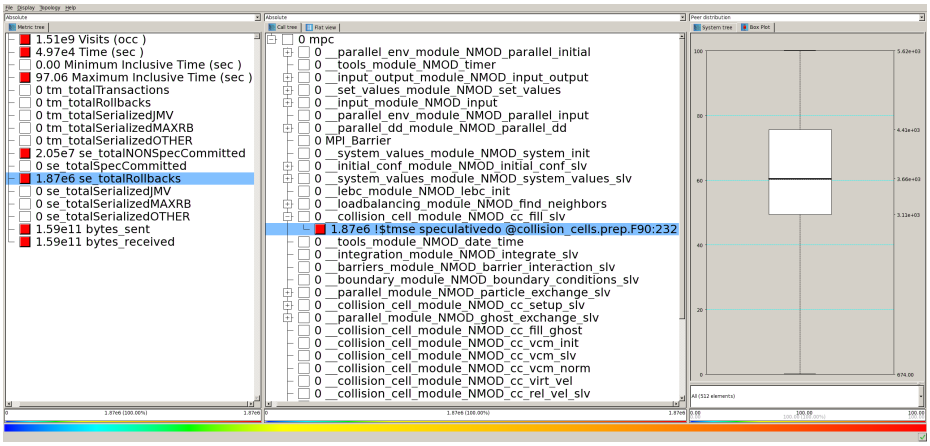

Fig. 3. Cube screenshot of the SE implementation of MP2C, showing the distribution of rollbacks among the processes as a boxplot.

this needs to be carefully balanced against the more limited information available for analysis. Our tool set will be continuously adapted to changes in the existing directive-based programming paradigms, in particular possible TM/SE support in the OpenMP specification. We further plan to integrate other models like OpenACC and Intel LEO. Adding support for tracing of TM/SE events – to be able to visualize these with Vampir [20] – will lead to deeper insights on specific instances of TM/SE regions. In addition, we will work on more sophisticated analysis for TM/SE, both for profiling and tracing. For example, it could be possible to color conflicting transactions in Vampir to see directly where rollbacks originate.

References

1. Ohmacht, M., Wang, A., Gooding, T., Nathanson, B., Nair, I., Janssen, G., Schaal, M., Steinmacher-Burow, B.: IBM Blue Gene/Q memory subsystem with speculative execution and transactional memory. IBM Journal of Research and Development **57**(1/2) (2013) 7–1
2. Knüpfer, A., et al.: Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Proc. of 5th Parallel Tools Workshop, 2011, Dresden, Germany, Springer (Sep 2012) 79–91
3. Geimer, M., Kuhlmann, B., Pulatova, F., Wolf, F., Wylie, B.J.N.: Scalable Collation and Presentation of Call-Path Profile Data with CUBE. In: Proc. of the Conference on Parallel Computing (ParCo), Aachen/Jülich, Germany. (Sep 2007) 645–652 *Minisymposium Scalability and Usability of HPC Programming Tools.*
4. Herlihy, M., Moss, J.E.B.: Transactional Memory: Architectural Support for Lock-free Data Structures. In: Proc. of the 20th Annual Intl. Symposium on Computer Architecture. ISCA '93, New York, NY, USA, ACM (1993) 289–300
5. Shavit, N., Touitou, D.: Software transactional memory. Distributed Computing **10**(2) (1997) 99–116

6. Ansari, M., Jarvis, K., Kotselidis, C., Luján, M., Kirkham, C., Watson, I.: Profiling transactional memory applications. In: Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on, IEEE (2009) 11–20
7. Zyulkyarov, F., Stipic, S., Harris, T., Unsal, O.S., Cristal, A., Hur, I., Valero, M.: Profiling and Optimizing Transactional Memory Applications. Intl. Journal of Parallel Programming **40**(1) (2012) 25–56
8. Lourenço, J., Dias, R., Luís, J., Rebelo, M., Pessanha, V.: Understanding the behavior of transactional memory applications. In: Proc. 7th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging, ACM (2009) 3
9. Cascaval, C., Blundell, C., Michael, M., Cain, H.W., Wu, P., Chiras, S., Chatterjee, S.: Software Transactional Memory: Why Is It Only a Research Toy? Queue **6**(5) (Sep 2008) 40:46–40:58
10. Wang, A., Gaudet, M., Wu, P., Amaral, J.N., Ohmacht, M., Barton, C., Silvera, R., Michael, M.: Evaluation of Blue Gene/Q hardware support for transactional memories. In: Proc. of the 21st international conference on Parallel architectures and compilation techniques, ACM (2012) 127–136
11. Schindewolf, M., Biliari, B., Gyllenhaal, J., Schulz, M., Wang, A., Karl, W.: What scientific applications can benefit from hardware transactional memory? In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, IEEE (2012) 1–11
12. Kunaseth, M., Kalia, R.K., Nakano, A., Vashishta, P., Richards, D.F., Glosli, J.N.: Performance Characteristics of Hardware Transactional Memory for Molecular Dynamics Application on BlueGene/Q: Toward Efficient Multithreading Strategies for Large-Scale Scientific Applications. In: Proc. of Intl. Workshop on Parallel and Distributed Scientific and Engineering Computing. (2013)
13. Schindewolf, M., Rocker, B., Karl, W., Heuveline, V.: Evaluation of Two Formulations of the Conjugate Gradients Method with Transactional Memory. In: Euro-Par 2013 Parallel Processing. Volume 8097 of LNCS. Springer (2013) 508–520
14. Bihari, B.L., Wong, M., Wang, A., Supinski, B.R., Chen, W.: A case for including transactions in openmp ii: Hardware transactional memory. In: OpenMP in a Heterogeneous World. Volume 7312 of LNCS. Springer (2012) 44–58
15. Yoo, R.M., Hughes, C.J., Lai, K., Rajwar, R.: Performance evaluation of Intel® transactional synchronization extensions for high-performance computing. In: Proc. of SC13: Intl. Conference for High Performance Computing, Networking, Storage and Analysis, ACM (2013) 19
16. `http://www.scalasca.org`
17. Mohr, B., Malony, A.D., Hoppe, H.C., Schlimbach, F., Haab, G., Hoeflinger, J., Shah, S.: A Performance Monitoring Interface for OpenMP. In: Proc. of Fourth European Workshop on OpenMP (EWOMP), Rome, Italy. (Sep 2002)
18. Maurer, T.: BG/Q Application Tuning – memory hierarchy, transactional memory, speculative execution. `http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/juqueenpt13/juqueenpt13-applicationtuning1.pdf`
19. Sutmann, G., Westphal, L., Bolten, M.: Particle based simulations of complex systems with mp2c: hydrodynamics and electrostatics. In: ICNAAM 2010: International Conference of Numerical Analysis and Applied Mathematics 2010. Volume 1281., AIP Publishing (2010) 1768–1772
20. Brunst, H., Mohr, B.: Performance Analysis of Large-Scale OpenMP and Hybrid MPI/OpenMP Applications with Vampir NG . In: OpenMP Shared Memory Parallel Programming. Volume 4315 of LNCS. Springer (2008) 5–14