The Importance of Run-time Error Detection

Glenn R. Luecke [1], James Coyle [1], James Hoekstra [1], Marina Kraeva [1], Ying Xu, Mi-Young Park, Elizabeth Kleiman, Olga Weiss, Andre Wehe, Melissa Yahya,

[1] Iowa State University's High Performance Computing Group,
Iowa State University, Ames, Iowa 50011, USA
{grl, jjc, hoekstra, kraeva}@iastate.edu

Abstract. The ability of system software to detect and issue error messages that help programmers quickly fix serial and parallel run-time errors is an important productivity criterion for developing and maintaining application programs. Over ten thousand run-time error tests and a run-time error detection (RTED) evaluation tool has been developed for the automatic evaluation of run-time error detection capabilities for serial errors and for parallel errors in MPI, OpenMP and UPC programs. Evaluation results, tests and the RTED evaluation tool are freely available at http://rted.public.iastate.edu. Many compilers, tools and run-time systems scored poorly on these tests. The authors make recommendations for providing better RTED in the future.

Keywords: Run-time error detection, Fortran, C, C++, MPI, OpenMP, UPC.

1 Introduction

Debugging serial and parallel programs can be very time consuming. The typical debugging process is: (1) first compile and correct all compile-time errors, (2) next run and correct the run-time errors issued by the run-time system and (3) then use a debugger and/or print statements to find and correct the rest of the errors, i.e. the errors not detected at run-time and logical errors. Compile-time errors can normally be corrected quickly since compilers usually issue good error messages. Similarly, usually run-time errors can be corrected quickly without a debugger and print statements when the run-time system correctly diagnoses the error and issues a good error message. However, correcting the other errors can be very time consuming. If a run-time system does not issue a good error message, then one is forced to use a debugger and/or print statements to find and correct the error. Notice that print statements and debuggers only give values of variables and it is up to the person debugging the program to know if these values are correct or not. Thus, high quality run-time error detection with high quality run-time error messages is critical to providing a productive computing environment. In addition, high quality run-time error detection would be especially valuable for petascale computing when conventional parallel debuggers may not scale to hundreds of thousands of cores.

The productivity enhancement from having excellent run-time error detection (RTED) depends on many factors; e.g., the type of error, the length and complexity of the program, the experience and intelligence of the person trying debug the program

as well as this person's knowledge of the program. A few years ago, an expert in parallel programming and professor of physics at Iowa State University (ISU) spent nine months trying to find what was causing the physics code that he wrote to abort after several days of execution. He then asked ISU's HPC Group to help find the error in his 6,000 line, Fortran-MPI application code. The code was run using the MPI-CHECK tool [12]. The error was detected and a good error message was issued. With this information, the professor was able to correct the error quickly. This physics code is now running on machines all over the world and is being run regularly with 30,000 processors. Without a run-time error detection tool, the error might have never been found.

With funding from the US Department of Defense, from DARPA's High Productivity Computing Initiative and from the Extreme Scale System Center at Oak Ridge National Laboratory, extensive run-time error tests have been developed from 2003 through 2008 by Iowa State University's High Performance Computing Group for evaluating run-time error detection capabilities for

- serial errors in programs written in Fortran, C and C++ (2695 tests)
- parallel MPI errors in programs written in Fortran, C and C++ (1942 tests)
- parallel OpenMP errors in programs written in Fortran, C and C++ (3307 tests)
- parallel programs written in Unified Parallel C (2247 tests).

More than ten thousand run-time error tests have been written for this project. A run-time error detection (RTED) evaluation tool has been developed for running these tests and for automatically evaluating the run-time error messages generated by assigning a score from 0 to 5 based on the usefulness of the message to help fix the error quickly. The tool then automatically averages these scores over the different error categories and reports the results. These tests and the RTED evaluation tool provide an easy way to evaluate and compare run-time error detection capabilities provided by different vendors and could be used as part of a computer procurement process. In addition, vendors could use these tests, recommended error messages and the RTED evaluation tool to evaluate and improve the run-time error detection capabilities of their compilers, tools and run-time systems.

Current test results, tests, desired output files, and the RTED evaluation tool are freely available at http://rted.public.iastate.edu. As new compilers, tools and run-time systems become available, vendors are encouraged to send the results using the new system software to rted.project@iastate.edu so they can be posted on this web site.

2 Background

There are many commercial and public domain software systems to detect and provide information to help programmers fix serial run-time errors. The survey in [1] found that the commercial products, Insure++ and Purify performed by far the best of all software systems evaluated. At the time of the study, Insure++ was considered better than Purify but both products did an excellent job in detecting serial run-time errors in C and C++ and provided excellent information for the quick fixing of the

errors. Unfortunately, neither Insure++ nor Purify find run-time errors for Fortran. Sun's HPC ClusterTools [2] contains the *bcheck* tool for finding serial run-time errors in Fortran, C and C++ programs.

The Message Passing Interface, MPI, is the standard message passing library used for writing parallel scientific programs for distributed memory parallel computers [3, 4]. OpenMP is often used for writing parallel scientific programs for shared memory parallel computers [5, 6]. Since most of today's parallel computers are a collection of shared memory compute nodes interconnected via a communication network, some application programs are written using both MPI and OpenMP and some using only MPI. Unified Parallel C [7, 8] is an extension of C for parallel execution on shared or distributed memory parallel machines. Some scientific applications are written entirely in UPC instead of MPI and/or OpenMP.

There are several tools to aid in the debugging of MPI programs. The Umpire tool [9, 10] was initially developed by Jeffrey Vetter and Bronis de Supinski in 2000 at Lawrence Livermore National Laboratory. The High Performance Computing Center (HLRS) in Stuttgart and the Technische Universitaet Dresden (ZIH) in Germany have developed the MARMOT tool [11] for finding MPI run-time errors in Fortran and C programs. Iowa State University's High Performance Computing Group has developed MPI-CHECK for Fortran [12]. Intel's *message checker* [13] is a tool that has been developed to find MPI run-time errors in Fortran, C and C++ programs. Intel has integrated *message checker* into their *trace analyzer* and *collector* 2.7 tools within their Cluster Toolkit.

Intel's *thread checker* [14] and Sun Microsystems' *thread analyzer* [2] are tools for debugging OpenMP run-time errors. The authors are not aware of any run-time error detection tools for Unified Parallel C (UPC).

3 Methodology

This section summarizes the methodology used to develop the run-time error tests and the software for the automatic running of tests and for the automatic evaluation of the error messages. For each run-time error, test programs have been written to determine if the error can be detected and a quality message generated (each test program contains one and only one run-time error). Tests were written so that the information needed to detect the error is not available at compile-time. For each test a file with a recommended error message was created that contains the error name, the line number and the file name where the error occurred along with any additional information that would assist a programmer to find and correct the error.

The following are the run-time error categories used for the development of the serial tests for Fortran, C and C++: array index out of bounds, uninitialized variables, subprogram call errors, pointer errors, floating point errors, string errors, allocation and deallocation errors, memory leaks, input and output errors, Fortran 95 specific errors, Fortran array conformance errors and C99 specific errors.

The following are the run-time error categories used for the development of the MPI tests for Fortran, C and C++: buffer out of bounds, buffer overlap, data type errors, rank errors, other argument errors, wrong order of MPI calls, negative message

length, deadlocks, race conditions, implementation dependent errors (potential deadlocks, race conditions and noncontiguous dynamic allocation of message buffers in C).

The run-time error categories used for the Fortran, C and C++ OpenMP tests are: deadlocks, race conditions, environment and clause errors, wrong order of OpenMP directives, uninitialized shared and private variables, wrong usage of OpenMP runtime library routines and implementation dependent errors (i.e. behavior is either undefined or said to be implementation dependent by the OpenMP API).

The run-time error categories used for UPC are: out-of-bounds shared memory access using indices, out-of-bounds shared memory access using pointers, out-of-bounds shared memory access in UPC functions, argument errors in UPC functions, wrong order of UPC calls, uninitialized variables, deadlocks, race conditions, memory related errors, undefined UPC operations, warnings. The "warnings" category includes tests where programmers should be warned of likely errors. For example, it makes no sense to have the "nelems" argument in reduction functions be zero even though the UPC specification allows this, so some tests have nelems = 0 and the RTED evaluation tool checks whether good warning messages are produced.

The RTED evaluation tool mentioned in Section 1 is a collection of scripts for the automatic running of tests, comparing actual messages with expected messages and then assigning a score of 0, 1, 2, 3, 4 or 5 to the message generated for each test:

- A score of 5 is given for a detailed error message that allows for the quick fixing of the error. For example, when there is an out-of-bounds access of the second dimension of array B, instead of issuing a message "out-of-bounds access in array B in line 1735 of file prog.f90", the message could mention that the problem occurred in the second dimension of B, the accessed value was 17 and the allocated range was from 1 to 16 and that B was allocated in line 923 of prog.f90.
- A score of 4 is given for error messages with more information than a score of 3 and less than 5. This is tailored for each test.
- A score of 3 is given for error messages with the correct error name, line number and the name of the file where the error occurred.
- A score of 2 is given for error messages with the correct error name and line number where error occurred but not the file name where the error occurred.
- A score of 1 is given for error messages with the correct error name.
- A score of 0 is given when the error was not detected.

Different compilers, tools and run-time systems may issue different messages (with different error names) for the same run-time error. For example, the out-of-bounds access is named as "array out-of-bounds error" on one system, "array index error" on another system and "index out-of-bounds error" on a third system. The RTED evaluation tool has a list of synonymous phrases for each error so that equivalent error messages will be evaluated appropriately. Thus, for the RTED tool to accurately evaluate an error message, the error name must be listed as one of the RTED synonymous phrases. Error messages are evaluated as follows:

- For each test and score a scoring script was created.
- A synonym file of acceptable error names was created.

- Error messages are reduced to a canonical form for easy comparison with the recommended error messages by first changing all text to lower case and then replacing selected phrases with standard phrases. Blanks, hex addresses, and integers longer than three digits are removed to reduce false matches.
- Scoring scripts are applied to the canonical form of each error message for automatic evaluation.

4 Results

This section contains the results of running all the tests using the software environments/machines that were available to us. Tests were run using all available compiler run-time error detection options. For each compiler, we searched the man pages and selected all debugging options for this evaluation. It would be helpful if compilers had a general "--debug" option that would turn on all debugging options. Results on the web site, http://rted.public.iastate.edu, present scores for each category of run-time errors. Due to space limitations, we cannot present all of the RTED results and only present average scores in most cases. Some vendors score well in some error categories and poorly in others. When taking averages, this information may be hidden, so the reader is encouraged to view the complete results posted on the web site.

There are 1552 serial execution Fortran tests, 716 serial execution C tests and 1143 serial execution C++ tests. (We ran the 716 C tests and 427 C++ specific tests for our 1143 C++ tests.) Table 1 presents the average scores when running these tests on different machines/software environments.

Notice that for the serial C and C++ tests, Insure++ is the only one that scored well. The Cray X1 and the NAG Fortran compilers both scored well.

Table 1. Average serial execution results.

Compiler/tool	Fortran	C	C++
Cray XT4 CNL, Pathscale compilers	0.62	0.08	0.06
Cray X2 CNL, Cray compilers	2.07	0.39	0.47
Cray X1 Unicos/mp, Cray compilers	2.49	0.47	0.53
Cray XT4 CNL, PGI compilers	1.37	0.43	0.28
IBM AIX, XLF/XLC compilers	1.51	0.12	0.10
RedHat Linux, NAGWare Fortran 95	2.36	NA	NA
RedHat Linux, Intel compilers	1.34	0.00	0.00
RedHat Linux, Intel compilers with Insure++	NA	2.75	2.97
SUN Solaris, Sun compilers with bcheck	2.11	1.08	1.29
SUN Solaris, Sun compilers	1.79	0.00	0.00
GNU v4.1.2 compilers	1.19	0.08	0.06

There are 744 MPI Fortran tests, 723 MPI C tests and 1198 MPI C++ tests. Table 2 presents the average scores when running the MPI 2.0 tests on different machines/software environments. The RTED web site contains the scores for both MPI 1.1 and for MPI 2.0, but only the MPI 2.0 results are presented in this paper. If

an MPI implementation does not support the full MPI 2.0 standard, there is no penalty when running the MPI 2.0 tests. This is because the RTED evaluation tool only assigns a score to those tests that compile and link successfully and average scores are calculated on the reduced set of tests. Notice that MPI-CHECK and Marmot scored better than the others. Intel's *trace analyzer* was not able to identify the line number where the error occurred so the best possible score for each test would be 1.0.

Table 2. Average MPI 2.0 results.

Compiler/MPI Library/tool	Fortran	C	C++
Cray X1 Unicos/mp, Cray compilers	0.64	0.75	0.82
Cray XT4 CNL, MPICH2, PGI compilers	0.24	0.32	0.31
IBM AIX, XLF/XLC compilers	0.40	0.46	0.43
RedHat Linux, OpenMPI, Pathscale	0.25	0.32	0.30
RedHat Linux, OpenMPI, Pathscale with MPI-CHECK	1.32	NA	NA
RedHat Linux, OpenMPI, Pathscale with Marmot	1.27	1.35	0.83
RedHat Linux, MPICH-gm, Intel compilers	0.25	0.46	0.43
Suse, MPICH 1.2, Intel compilers with Trace Analyzer	0.47	0.50	0.48
SUN Solaris, MPICH2, Sun compilers	0.29	0.34	0.32

Since MPI is the most commonly used method of parallelization, table 3 presents the detailed MPI 2.0 results for Fortran and for selected compilers, MPI Libraries and tools. Cray XT results are for the Cray XT4 system, using MPICH2 library and PGI Fortran compiler. Intel TA results were obtained on a Xeon cluster running MPICH 1.2 using Intel compiler along with the Intel Trace Analyzer and Collector tool. OpenMPI results are for an Opteron cluster, using OpenMPI library and Pathscale Fortran compiler. The results in the last two columns were obtained on the same Opteron cluster, using the MPI-CHECK and Marmot tools.

Table 3. MPI 2.0 Fortran results for each error category.

MPI Error Category	Cray XT	Intel TA	OpenMPI	MPI-CHECK	Marmot
Buffer out of bounds	0.01	0.06	0.02	2.13	0.05
Buffer overlap	0.00	0.71	0.05	0.62	0.14
Datatype errors	0.10	0.40	0.20	1.80	0.20
Rank errors	0.54	0.76	0.08	2.81	1.87
Other argument errors	0.22	0.59	0.00	0.30	0.37
Wrong order of MPI calls	0.48	0.69	0.06	0.56	1.48
Negative message length	0.98	0.91	0.08	1.95	1.42
Deadlocks	0.02	0.17	0.00	1.79	2.69
Race conditions	0.00	0.06	0.00	0.09	1.07
Implementation dependent errors	0.04	0.31	0.00	3.18	3.39
AVERAGES	0.24	0.47	0.05	1.52	1.27

The following is one of the Fortran MPI tests that we wrote:

```
41  program F_C_1_3_1_2_e_M1
43    implicit none
44    include "mpif.h"
46    integer, parameter :: N=5     ! buffer size
48    integer :: arrayA(N), arrayB(N+1), sbuf(N+1)
55    call MPI_INIT(ierror)
56    call MPI_COMM_SIZE(mpi_comm_world, numprocs,
                         ierror)
57    call MPI_COMM_RANK(mpi_comm_world, myrank,
                         ierror)
65    if(cos(x) > 2.0) then
66       count = N
67    else
68       count = N+1
69    endif
70
71    do i=1,count
72       sbuf(i) = myrank + i
73    enddo
74
75    if(myrank.eq.1) then
76       call MPI_ALLREDUCE(sbuf, arrayA, count,
          MPI_INTEGER, MPI_SUM, mpi_comm_world, ierror)
77    else
78       call MPI_ALLREDUCE(sbuf, arrayB, count,
          MPI_INTEGER, MPI_SUM, mpi_comm_world, ierror)
79    endif
87    call MPI_FINALIZE(ierror)
91  end program F_C_1_3_1_2_e_M1
```

This program reads the value of x from a file so that its value is not known to the program at compile time. For this example Pathscale with OpenMPI, Pathscale with OpenMPI and Marmot, and Intel's Trace Analyzer and Collector all did not detect the error and were given a score of zero. MPI-CHECK with the Pathscale compiler and OpenMPI received a score of 4.0 for producing the following message:

> File=/scratch/jjc/F_C_1_3_1_2_e_M1.f90, Line= 76 , Argument= 2] arrayA, message size exceeds the bounds of this array, please check the message size.

The recommended error message for this test is:

> Buffer size exceeded. The value 6 of argument 'count' in 'MPI_ALLREDUCE' called at line 76 in file 'F_C_1_3_1_2_e_M1.f90' on process 1 exceeds the size of receive buffer 'arrayA'. 'arrayA' is declared at line 48 in file 'F_C_1_3_1_2_e_M1.f90' with size 5.

There are 2156 OpenMP Fortran tests, 1066 OpenMP C tests, and 1151 OpenMP C++ tests. (We ran the 1066 C tests and 85 C++ specific tests for our 1151 C++

tests.) There are few C++ specific tests since the OpenMP API has few items that are C++ specific. Table 4 presents the average scores when running these tests on different machines/software environments. Notice that Intel's *thread checker* and Sun's *thread analyzer* both improved the score, but not by much.

Table 4. Average OpenMP results.

Compiler/tool	Fortran	C	C++
Cray X1 Unicos(mp), Cray compilers	0.32	0.30	0.45
Cray X2 Unicos(mp), Cray compilers	0.23	0.25	0.40
Cray Unicos, PGI compilers	0.17	0.19	0.21
Cray Unicos, GNU compilers	0.20	0.19	0.27
Cray Unicos, Pathscale compilers	0.13	0.18	0.21
IBM AIX, XLF/XLC compilers	0.26	0.23	0.30
RedHat Linux, Pathscale compilers	0.13	0.19	0.24
RedHat Linux, Intel compilers	0.13	0.14	0.20
RedHat Linux, Intel compilers with thread checker	0.42	0.43	0.52
SUN Solaris, Sun compilers	0.02	0.02	0.03
SUN Solaris, Sun compilers with thread analyzer	0.40	0.39	0.40

Table 4 presents the scores when running the 2247 UPC tests using Cray's, Berkeley's, HP's and GNU's UPC compilers. The section "Undefined UPC Operations" contains all situations where the outcome of certain UPC statements is stated as being undefined by the UPC specification. The UPC "warnings" category is described in section 3. GNU's and HP's UPC do not support UPC IO so these tests were skipped when using these compilers and scores calculated on the reduced set of tests. In addition, GNU's UPC does not support UPC collective utilities so these tests were also skipped when using GNU's UPC compiler and scores calculated on the reduced set of tests. Notice that Cray's UPC compiler scored better than all the others in some categories.

Table 5. UPC results for each error category.

UPC Error Category	Cray	Berkeley	HP	GNU
Out-of-bounds shared memory access using indices	1.30	0.00	0.03	0.20
Out-of-bounds shared memory access using pointers	1.04	0.00	0.00	0.21
Out-of-bounds shared memory access in UPC calls	0.91	0.00	0.02	0.01
Argument errors in UPC functions	0.38	0.04	0.00	0.00
Wrong order of UPC calls	0.84	0.20	0.53	0.89
Uninitialized variables	0.08	0.02	0.57	0.25
Deadlocks	0.00	0.58	0.36	0.27
Race conditions	0.01	0.00	0.00	0.00
Memory related errors	0.18	0.00	0.16	0.37
Undefined UPC operations	0.19	0.21	0.15	0.41
Warnings (uninitialized shared variables)	0.27	0.00	0.00	0.00
AVERAGES	0.47	0.10	0.17	0.24

5 Recommendations

The ability to detect and issue high quality run-time error messages is critical for programmer productivity and should be an integral part of providing a productive environment for the development and maintenance of scientific applications. In this section, we make recommendations on how this could be accomplished.

Ideally, each vendor should provide high-quality RTED. However, the results of this study show that this is not the current situation and there are no signs that this will change. Since JAVA's language specification includes array bounds checking, we thought that RTED could be part of the Fortran, C and C++ language specifications as a first step towards providing high quality RTED. Vendors could then use the RTED tests developed in the project when implementing RTED. This idea was presented to the Fortran standards committee. After discussion the idea was rejected. The idea has not been presented to the C and C++ standards committees.

Since high quality RTED is so important for the productivity of application code developers, we now recommend the development of high quality RTED tools that are freely available and support each commonly-used programming paradigm. Funding for these tools must include not only their development but also ongoing maintenance, periodic enhancement with better RTED techniques and with programming paradigm advancements. The following lists the specific programming paradigms we recommend RTED tools be developed for:

- Serial Fortran, C, C++
- MPI with Fortran, C and C++
- OpenMP with Fortran, C and C++
- UPC
- Co-Array Fortran

Since MPI programs are Fortran, C or C++ programs calling MPI functions, the MPI tool should be used along with the serial tools for Fortran, C and C++. The OpenMP tools could be developed from the serial Fortran, C and C++ tools.

Since NVIDIA is providing CUDA for programming for their GPU, we recommend NVIDIA fund the development and maintenance of RTED tools for CUDA.

There are three new parallel languages that have been developed as part of DARPA's High Productivity Computing Initiative: Chapel developed by Cray, Fortress developed by Sun Microsystems and Q10 developed by IBM. We recommend that each of these vendors fund the development and maintenance of a high quality RTED tool for their own language.

6 Conclusions

The ability of system software to detect errors at run-time and issue error messages that help programmers quickly fix errors is an important productivity criterion for developing and maintaining application programs. Over ten thousand run-time error tests and a run-time error detection (RTED) evaluation tool have been developed for the automatic evaluation of run-time error detection capabilities for serial errors and

for parallel errors in MPI, OpenMP and UPC programs. Each error message issued by the run-time system is assigned a score from 0 to 5 based on the usefulness of the information in the message to help a programmer quickly fix the error. Average scores over error categories are automatically calculated and reported. All tests and the RTED evaluation tool are freely available at the RTED web site http://rted.public.iastate.edu. Many compilers, tools and run-time systems have been evaluated with results posted on this same web site.

The technology for detecting and reporting many run-time errors is known, but the results of running these tests show that many of the software environments evaluated currently do a poor job detecting run-time errors with the following exceptions:

- For the serial tests, Insure++ scored well for C and C++ programs and the Cray X1 and NAG compilers both scored well for Fortran.
- For the MPI tests, MPI-CHECK and Marmot scored better than the others.

It is hoped that these tests and recommended error messages will be used to evaluate and improve the run-time error detection capabilities of compilers, tools and run-time systems and that these tests will also be used by high performance computing centers as part of their computer procurement process.

The authors recommend the development of high-quality, public domain RTED tools to support the programming paradigms commonly used for scientific computing. Funding for these projects should include not only development but also maintenance, periodic enhancements with better RTED techniques and support future programming paradigm enhancements.

Acknowledgments. This work was funded by the US Department of Defense, DARPA's High Productivity Computing Initiative and by the Extreme Scale System Center at Oak Ridge National Laboratory.

References

1. Luecke, G., Coyle, J., Hoekstra, J., Kraeva, M., Li, Y., Taborskaia, O., Wang, Y.: A Survey of Systems for Detecting Serial Run-time Errors. Concurrency and Computation: Practice and Experience, vol. 18, pp 1885--1907 (2006)
2. Sun Microsystem's HPC ClusterTools, http://www.sun.com/software/products/clustertools
3. Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., Dongarra, J.: MPI - The Complete Reference, The MIT Press (1998)
4. Message Passing Interface Forum, http://www.mpi-forum.org
5. The OpenMP API Specification, http://openmp.org
6. Chapman, B., Jost, G., Van der Pas, R.: Using OpenMP: Portable Shared Memory Parallel Programming, The MIT Press (2008)
7. Unified Parallel C, http://upc.gwu.edu
8. El-Ghazawi, T., Carlson, W., Sterling, T., Yelick, K.: UPC Distributed Shared Memory Programming, Wiley-Interscience (2005)
9. Vetter, J.S., De Supinski, B.R.: Dynamic software testing of MPI applications with Umpire, In: Conference on High Performance Networking and Computing Article 51, Proceedings of the 2000 ACM/IEEE conference on Supercomputing, Dallas, Texas, United States (2000)

10. Hilbrich, T., Supinski, B., Mueller, M., Schulz, M.: A Graph Based Approach for MPI Deadlock Detection, In: International Conference on Supercomputing, Yorktown Heights, NY, USA, pp 296--305 (2009)
11. MARMOT, http://www.hlrs.de/organization/av/amt/research/marmot/publications
12. Luecke, G.R., Chen, H., Coyle, J., Hoekstra, J., Kraeva, Zou, Y.: MPI-CHECK: a Tool for Checking Fortran 90 MPI Programs. Concurrency and Computation: Practice and Experience, vol. 15, pp 93--100 (2003)
13. Intel Message Checker, http://www.intel.com/cd/software/products/asmo-na/eng/227074.htm
14. Intel Thread Checker, http://software.intel.com/en-us/intel-thread-checker/