UPC-CompilerCheck: A Tool for Evaluating Error Detection Capabilities of UPC Compilers

Marina Kraeva[†], James Coyle[‡], Glenn R. Luecke[*], Indranil Roy[§], Elizabeth Kleiman[‖], and James Hoekstra[¶]
High Performance Computing Group, Iowa State University,
Ames, Iowa 50011, USA
Email: [†]kraeva@iastate.edu, [‡]jjc@iastate.edu, [*]grl@iastate.edu,
[§]iroy@iastate.edu, [‖]ekleiman@mtmercy.edu and [¶]hoekstra@iastate.edu

***Abstract*—The ability of system software to detect compile-time errors and issue messages that help programmers quickly fix these errors is an important productivity criterion for developing and maintaining application programs. To evaluate this capability for Unified Parallel C (UPC) compilers, 3141 Compile-Time Error Detection (CTED) tests and a CTED evaluation tool, called UPC-CompilerCheck, have been developed. UPC-CompilerCheck assigns a score from 0 to 5 for each compiler-generated error message based on the usefulness of the information in the message to help a programmer fix the error quickly. This tool also calculates average scores for each error category and then prints the results. Compiler vendors could use UPC-CompilerCheck to evaluate and improve the compile-time error detection capabilities of their UPC compilers. All tests, UPC-CompilerCheck and test results for the Berkeley, Cray, GNU and HP UPC compilers are freely available at http://hpcgroup.public.iastate.edu/CTED/UPC.**

***Keywords*-Languages; UPC; compile-time error detection.**

I. Introduction

Unified Parallel C (UPC) is an extension of the C programming language for parallel execution on shared and distributed memory parallel machines [1], [2]. UPC uses a single shared, partitioned address space, where shared variables may be directly read and written by any thread. Shared variables are stored in the memory of the thread for which they have affinity. "UPC combines the programmability advantages of the shared memory programming paradigm and the control over data layout and performance of the message passing programming paradigm" [3]. Providing a productive programming environment for UPC will encourage new scientific applications to be written in UPC. Since debugging UPC programs can be time consuming, it is important to have UPC compilers, tools and run-time systems that can detect both compile-time and run-time errors and issue messages that help programmers quickly fix the errors. A tool to evaluate error detection capabilities of UPC run-time systems has already been developed [4]. This paper describes the UPC-CompilerCheck tool for evaluating error detection capabilities of UPC compilers.

Application programs are usually developed by (a) writing the application, (b) compiling the application and fixing all errors detected at compile time, (c) running the application and fixing all errors detected at run-time and (d) then validating the program using problems for which answers are known. Compile-time tools cannot be expected to find all errors, so run-time error detection tools such as [5], [6] will often be needed. However, when errors can be found at compile-time, programmer productivity will be increased.

To evaluate error detection capabilities of UPC compilers, 3141 UPC compile-time error detection (CTED) tests and the UPC-CompileCheck tool have been developed by ISU's HPC Group. Each test contains exactly one UPC compile-time error. The UPC-CompileCheck tool compiles these tests, assigns a score from 0 to 5 based on the quality of the error message, calculates the averages of these scores for each error category and reports results.

These tests and UPC-CompilerCheck provide an easy way to evaluate and compare compile-time error detection capabilities of different UPC compilers and could be used as part of a computer procurement process along with the UPC RTED tests [4]. In addition, compiler vendors could use the CTED tests, recommended error messages and UPC-CompilerCheck to evaluate and improve the compile-time error detection capabilities of their UPC compilers.

UPC compile-time tests, recommended error messages, UPC-CompilerCheck and test results are freely available at http://hpcgroup.public.iastate.edu/CTED/UPC. As new UPC compilers/releases become available, vendors and researchers are encouraged to send results to cted.project@iastate.edu so that they can be posted on this web site.

II. Background

The UPC Compiler Group at the University of California Berkeley/Lawrence Berkeley National Laboratory actively participated in writing of the UPC Specification and developed the first UPC compiler. This compiler is an open-source and portable implementation of UPC [3]. Cray, HP, GNU and IBM have also developed UPC compilers.

The UPC working group at the High Performance Computing Lab (HPCL) at George Washington University is involved in the UPC specification, UPC testing strategies, UPC

documentation, UPC testing suites, UPC benchmarking, and UPC collective and Parallel I/O specification [7].

At Michigan Technological University work on UPC includes the recent release of the MuPC run-time system for UPC as well as collective specification development, memory model research, programmability studies, and test suite development [8].

Researchers at the University of Florida's High Performance Computing and Simulation Laboratory are currently involved in the research and development of a next-generation performance analysis tool supporting UPC. This tool helps users to identify bottlenecks in their programs and serves as a test-bed for advanced analysis techniques aimed at increasing programmer productivity [9].

The High Performance Computing Group from Iowa State University has developed a run-time error detection tool called UPC-CHECK that includes deadlock detection [5], [10]. In addition the ROSE-CIRM tool [6] has been developed by Dan Quinlan's group at Lawrence Livermore National Laboratory to complement UPC-CHECK by detecting other run-time errors. Ali Ebnenasir from the Department of Computer Science of Michigan Technological University developed UPC-SPIN [11], a software framework for the model checking of the inter-thread synchronization functionalities of Unified Parallel C (UPC) programs. A list of UPC programming tools can be found in the Programming Tools section of the UPC Wiki page [2].

III. Methodology

This section summarizes the methodology used to develop compile-time error tests and UPC-CompilerCheck. For each error, a program has been written that contains the specified error and no other errors (each program contains one and only one compile-time error). 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 occurs along with any additional information that would assist a programmer to find and correct the error.

The UPC compile-time error tests have been written to cover a wide range of errors in many different situations. The following are the UPC compile-time error categories:

- items that the UPC specification explicitly does not allow and that should be detected at compile-time
- out-of-bounds shared memory access using indices
- out-of-bounds shared memory access using pointer references
- out-of-bounds shared memory access in UPC library functions
- argument errors in UPC library functions
- wrong order of UPC statements and function calls
- uninitialized variables
- deadlocks
- race conditions
- memory leaks and memory related errors
- operations specifically undefined by the UPC specification
- warnings

The UPC CTED evaluation tool is a collection of scripts for compiling the tests, comparing actual messages with expected messages and then assigning a score of 0, 1, 2, 3, 4 or 5 to the message generated by each test. Scores for messages are assigned as follows:

- A score of 0 is given when the error is not detected.
- A score of 1 is given for error messages with the correct error name.
- A score of 2 is given for error messages with the correct error name and line number where the error occurred but not the file name where the error occurred.
- 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 4 is given for error messages which contain at least the information required for a score of 3 but less information than needed for a score of 5.
- A score of 5 is given in all cases when the error message contains all the information needed for fixing the error quickly.

The scoring is the same as was done for the run-time tests [4] even though for compile-time tests if a compiler identifies the correct line number it is likely that it also will identify the correct file name. This means for compile-time tests that the score of 2 will likely never be given. The information needed for scores of 4 and 5 is tailored to each test. Examples in Section IV illustrate this.

Different compilers may issue different messages (with different error names) for the same compile-time error. UPC-CompilerCheck has a list of synonymous phrases for each error so that equivalent error messages will be evaluated appropriately. Additional synonymous phrases may need to be added as new compilers/releases become available. Error messages were evaluated by UPC-CompilerCheck as follows:

- For each test and score, a scoring script was created.
- Error messages were 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, hexadecimal addresses, and integers longer than three digits are removed to reduce false matches.
- Scoring scripts were applied to the canonical form of each error message for evaluation.

UPC-CompilerCheck has been designed for easy usage. To run all tests one sets up the configuration file and then issues the 'run_tests_all' command. Sample configuration files for each compiler are provided. By issuing the 'run_tests_all <error_category>' command the user can run only the tests in the selected error category. The 'run_tests_all' command

also calculates average scores for each error category and then prints the results. UPC-CompilerCheck also allows one to run individual tests and to examine the output.

IV. Examples

This section contains four examples to illustrate how the tests have been written and how messages were scored.

A. Example 1

Applying a binary operator with incorrect operands.

Example 1: **c_A_3_1_a_A.upc**

```
...
26 #include "upcparam.h"
27 #include <stddef.h>
28
29 shared [4] char Arr_A[4*THREADS];
30
31 int main() {
32      shared [4] char *Ptr_S;
33      char *Ptr_L;
34      ptrdiff_t diff;
35
36      Ptr_S=&Arr_A[4*MYTHREAD+1];
37      Ptr_L=(char *)&Arr_A[4*MYTHREAD];
38
39      diff=Ptr_S-Ptr_L;
40
41      if(MYTHREAD==0) {
42           printf("diff = %d\n",diff);
43      }
44
45      return 0;
46 }
```

The following is the recommended error message:

```
ERROR: incorrect operands
An attempt to apply subtraction binary
operator to pointer-to-shared 'Ptr_S'
and pointer-to-local 'Ptr_L' is made at
line 39 in file 'c_A_3_1_a_A.upc'.
The pointer 'Ptr_S' is declared at line
32 in file 'c_A_3_1_a_A.upc'.
The pointer 'Ptr_L' is declared at line
33 in file 'c_A_3_1_a_A.upc'.
```

A score of 3 was given to the Berkeley UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
c_A_3_1_a_A.upc: In function 'main':
c_A_3_1_a_A.upc:39: warning: Attempt to
take the difference of pointer-to-shared
and pointer-to-private
```

A score of 3 was given to the GNU UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
c_A_3_1_a_A.upc: In function â:
c_A_3_1_a_A.upc:39: error: Attempt
to take the difference of shared and
nonshared pointers
```

A score of 0 was given to the Cray and HP UPC compilers since they did not detect the error.

B. Example 2

Using an uninitialized pointer.

Example 2: **c_H_2_l.upc**

```
. . .
25 #include "upcparam.h"
26 #define N 10
27
28 int main() {
29      shared double *ptr_x;
30      double* ptr_x1;
31
32      if(MYTHREAD==THREADS/2) {
33           ptr_x1=(double*)ptr_x;
34           ptr_x=(shared double*) upc_alloc(N*sizeof(double));
35           ptr_x1--;
36           printf("ptr_x=%p; ptr_x1=%p \n", (double*) ptr_x,
                  (double*) ptr_x1);
37
38           upc_free(ptr_x);
39      }
40
41      return 0;
42 }
```

The following is the recommended error message:

```
ERROR: uninitialized pointer
An attempt to assign pointer 'ptr_x'
that is not explicitly initialized to
another pointer is made at line 33 in
file 'c_H_2_l.upc'.
The pointer 'ptr_x' is declared at line
29 in file 'c_H_2_l.upc'.
```

A score of 4 was given to the Cray UPC compiler for issuing the following message since it correctly identified the error, file name, line number and gave the variable name. It was not given a score of 5 since the message did not give the line number where ptr_x was declared.

```
CC-7212 cc: WARNING File = c_H_2_l.upc,
Line = 33
Variable ``ptr_x'' is used before it is
defined.
```

A score of 4 was given to the HP UPC compiler for issuing the following message since it correctly identified the error, file name, line number and gave the variable name. It was not given a score of 5 since the message did not give the line number where ptr_x was declared.

```
``c_H_2_1.upc'', line 33: warning:
variable ``ptr_x'' is used before its
value is set.
ptr_x1=(double*)ptr_x;
                ^
```

A score of 0 was given to the Berkeley and GNU UPC compilers since they did not detect the error.

C. Example 3

An out-of-bounds array access error.

Example 3: **c_D_1_d_E.upc**

```
. . .
25 #define N 40
26 #define M 45
27
28 shared [] long double arrA[N]; /*DECLARE1*/
29 int main() {
30      long double var_res;
31      int i;
32
33      upc_forall(i=0;i<N;i++;&arrA[i])
34      arrA[i] = (long double)(i+1);
35      var_res = 10;
36      upc_barrier;
37
38      if(MYTHREAD == (THREADS-1)){
39      /*ERROR*/
40           arrA[N-M] = var_res;
41           for(i=0;i<N;i++)
42                printf("arrA[%d]=%d\n", i, (int)arrA[i]);
43      }
44
45      return 0;
46 }
```

The following is the recommended error message:

```
ERROR: out of bounds
Index value -5 is out of bounds for
array 'arrA' at line 40 in file
'c_D_1_d_E.upc'.
The array 'arrA' is declared with bounds
0:39 at line 28 in file 'c_D_1_d_E.upc'.
```

A score of 3 was given to the HP UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
``c_D_1_d_E.upc'', line 40: warning:
subscript out of range
arrA[N-M] = var_res;
     ^
```

A score of 3 was given to the Cray UPC compiler for issuing the following message since it correctly identified the error, file name and line number.

```
CC-175 cc: WARNING File = c_D_1_d_E.upc,
Line = 40
The indicated subscript is out of range.
arrA[N-M] = var_res;
     ^
```

A score of 0 was given to the Berkeley and GNU UPC compilers since they did not detect the error.

D. Example 4

An array declarator error when compiled with the dynamic threads environment option.

Example 4: **c_A_4_3_b.upc**

```
. . .
24 #include "upcparam.h"
25
26 #define SIZE 10
27
28 shared [2] int Arr_A[SIZE];
29
30 int main() {
31      int i;
32
33      if(MYTHREAD==0) {
34           for(i=0;i<SIZE;i++)
35                Arr_A[i]=SIZE+i;
36           printf("Arr_A[0]=%d\n", Arr_A[0]);
37      }
38
39      return 0;
40 }
```

The following is the recommended error message:

```
ERROR: invalid array declarator
THREADS is not used in the array
'Arr_A' declaration at line 28 in file
'c_A_4_3_b.upc'.
In the dynamic translation environment,
THREADS must appear exactly once in
declarations of shared arrays with
definite block size, either alone
or multiplied by an integer constant
expression.
```

The Cray UPC compiler was given a score of 5 since it contains all the information in the recommended error message. The Cray UPC compiler issued the following message:

```
CC-1560 cc: ERROR File = c_A_4_3_b.upc,
Line = 28
One dimension of an array of a shared
type must be a multiple of THREADS when
the number of threads is nonconstant.
shared [2] int Arr_A[SIZE];
                   ^
```

The Berkeley UPC compiler was also given a score of 5 for issuing the following message:

```
upcc: error during UPC-to-C translation
```

```
(sgiupc stage):
c_A_4_3_b.upc:28: In the dynamic
translation environment, THREADS must
appear exactly once in declarations of
shared arrays with definite block size.
Offending variable: Arr_A
```

The GNU UPC compiler was given a score of 3 for issuing the following message:

```
c_A_4_3_b.upc:28: error: variable-size
type declared outside of any function
c_A_4_3_b.upc:28: error: variable-size
type declared outside of any function
```

The HP UPC compiler was given a score of 5 for issuing the following message:

```
``c_A_4_3_b.upc'', line 28: error: one
dimension of an array of a shared type
must be a multiple of THREADS when the
number of threads is nonconstant
shared [2] int Arr_A[SIZE];
                    ^
```

V. Results

Table I presents the average scores for each error category when compiling the UPC CTED tests using the Berkeley, Cray, GNU and HP UPC compilers. Authors were not able to get access to the IBM UPC compiler. Current results are listed on the web site http://hpcgroup.public.iastate.edu/CTED/UPC.

The category "explicitly disallowed statements" contains items that the UPC specification explicitly does not allow and that should be detected at compile-time. The category "undefined UPC operations" contains situations where the outcome of certain UPC statements is stated as being undefined by the UPC specification. The "warnings" category includes tests where programmers should be warned of likely errors, e.g. use of deprecated functions, shared variables not initialized by the program, etc.. The "argument errors in UPC library functions" category covers those situations where inconsistent and/or incorrect information is passed as arguments to UPC library functions. At the time this project was done, the GNU and HP UPC compilers did not support UPC I/O so they scored zero on these tests. Notice from Table I that all the compilers achieved an average score of nearly 3.0 in the "explicitly disallowed statements" category.

For an error message to be useful it should receive at least a score of 3. Table II presents the number of tests in each error category for which an useful error message was issued. Together Tables I and II show that not only the quality of the error messages issued is low but also that the UPC compilers tested are not able to detect many of the errors.

Error category	**Berkeley**	**Cray**	**GNU**	**HP**
explicitly disallowed statements	2.92	2.75	3.21	2.62
out-of-bounds shared memory access using indices	0.00	1.00	0.00	1.27
out-of-bounds shared memory access using pointers	0.00	0.00	0.00	0.25
out-of-bounds shared memory access in UPC function calls	0.00	0.00	0.00	0.00
argument errors in UPC functions	0.00	0.07	0.05	0.09
wrong order of UPC statements and function calls	0.00	0.15	0.00	0.10
uninitialized variables	0.00	1.14	0.00	2.29
deadlocks	0.00	0.00	0.00	0.00
race conditions	0.00	0.00	0.00	0.00
memory related errors	0.00	0.00	0.00	0.09
undefined UPC operations	0.16	0.21	0.16	0.21
warnings	0.00	0.00	0.00	1.84
average of the above scores	**0.28**	**0.48**	**0.34**	**0.73**

Table I
AVERAGE OF TEST SCORES FOR EACH ERROR CATEGORY AND THE AVERAGE SCORE OVER ALL ERROR CATEGORIES FOR EACH UPC COMPILER.

Error Category	**Number of tests**	**Berkeley**	**Cray**	**GNU**	**HP**
explicitly disallowed statements	164	133	103	149	98
out-of-bounds shared memory access using indices	462	0	154	0	196
out-of-bounds shared memory access using pointers	169	0	0	0	14
out-of-bounds shared memory access in UPC function calls	324	0	0	0	0
argument errors in UPC functions	284	0	5	5	6
wrong order of UPC statements and function calls	158	0	6	0	4
uninitialized variables	35	0	10	0	20
deadlocks	18	0	0	0	0
race conditions	785	0	0	0	0
memory related errors	644	0	0	0	14
undefined UPC operations	19	1	1	1	1
warnings	79	0	0	0	29

Table II
NUMBER OF TESTS IN EACH CATEGORY WHICH RECEIVED A SCORE OF AT LEAST 3

The authors consider that all tests from the "explicitly disallowed statements" category should receive at least a score of 3. Table III shows in detail how the compilers scored on these tests including the number of tests that received a score of at least 3. Notice that many of these errors were not recognized by UPC compilers. The total number of tests in this category is 164.

Score	Berkeley	Cray	GNU	HP
0	31	61	15	66
1	0	0	0	0
2	0	0	0	0
3	90	7	106	5
4	6	50	6	51
5	37	46	37	42
3-5	**133 (81.1%)**	**103 (62.8%)**	**149 (90.1%)**	**98 (59.8%)**

Table III
NUMBER OF TESTS OUT OF 164 RECEIVING THE INDICATED SCORE FOR THE "EXPLICITLY DISALLOWED STATEMENTS" CATEGORY.

VI. CONCLUSION

The ability of system software to detect compile-time errors and issue messages that help programmers quickly fix these errors is an important productivity criterion for developing and maintaining application programs. To evaluate this capability for Unified Parallel C (UPC), 3141 compile-time error tests and a UPC-CompileCheck tool have been developed. For each error message issued, UPC-CompilerCheck assigns a score from 0 to 5 based on the usefulness of the information in the message to help a programmer quickly fix the error. If no error message is issued the test gets score of 0. UPC-CompilerCheck calculates average scores over each error category and then prints the results. All tests and UPC-CompilerCheck are freely available at http://hpcgroup.public.iastate.edu/CTED/UPC.

The Berkeley, Cray, GNU and HP UPC compilers have been evaluated and results posted on this same web site. Error detection capabilities for these compilers were generally poor including the error category "explicitly disallowed statements" where the UPC compilers should have detected all these errors.

It is hoped that these tests and recommended error messages will be used by vendors to evaluate and improve the compile-time error detection capabilities of their UPC compilers. We also hope that these tests will be used by high performance computing centers as part of their procurement process to reward vendors whose UPC implementations provide excellent compile-time (and run-time) error detection and issue high quality messages.

ACKNOWLEDGMENT

This work was supported by the United States Department of Defense and used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.

REFERENCES

[1] T. El-Ghazawi, W. Carlson, T. Sterling, and K. Yelick, *UPC: Distributed Shared Memory Programming*. Wiley-Interscience, 2003.

[2] "Unified parallel C (upc wiki)," last accessed April 27, 2012. [Online]. Available: http://upc.wikinet.org

[3] "The Berkeley Unified Parallel C," last accessed April 27, 2012. [Online]. Available: http://upc.lbl.gov/

[4] G. R. Luecke, J. Coyle, J. Hoekstra, M. Kraeva, Y. Xu, E. Kleiman, and O. Weiss, "Evaluating error detection capabilities of UPC run-time systems," in *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*, ser. PGAS '09. New York, NY, USA: ACM, 2009, pp. 7:1–7:4. [Online]. Available: http://doi.acm.org/10.1145/1809961.1809971

[5] J. Coyle, I. Roy, M. Kraeva, and G. R. Luecke, "UPC-CHECK: A scalable tool for detecting run-time errors in Unified Parallel C," in *Proceedings of International Supercomputing Conference (ICS)*, June 2012, to appear.

[6] P. Pirkelbauer, C. Liao, T. Panas, and D. Quinlan, "Runtime detection of c-style errors in upc code," in *Proceedings of Fifth Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '11, 2011. [Online]. Available: http://pgas11.rice.edu/papers/PirkelbauerEtAl-UPC-Error-Detect-PGAS11.pdf

[7] "The High Performance Computing Laboratory, The George Washington University," last accessed April 27, 2012. [Online]. Available: http://upc.gwu.edu

[8] "UPC projects at Michigan Technological University," last accessed April 27, 2012. [Online]. Available: http://www.upc.mtu.edu/

[9] "High Performance Computing and Simulation Laboratory, University of Florida," last accessed April 27, 2012. [Online]. Available: http://www.hcs.ufl.edu/upc/

[10] I. Roy, G. R. Luecke, J. Coyle, and M. Kraeva, "An optimal deadlock detection algorithm for Unified Parallel C," preprint (2012). [Online]. Available: http://hpcgroup.public.iastate.edu/papers/Deadlock_Dectection_for_UPC.pdf

[11] A. Ebnenasir, "UPC-SPIN: A Framework for the Model Checking of UPC Programs," in *Proceedings of Fifth Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '11, 2011. [Online]. Available: http://pgas11.rice.edu/papers/Ebnenasir-UPC-Model-Checking-PGAS11.pdf