Evaluating Error Detection Capabilities of UPC Compilers

by
Glenn R. Luecke, James Coyle, James Hoekstra, Marina Kraeva,
Elizabeth Kleiman, Indranil Roy
Iowa State University's High Performance Computing Group
Iowa State University, Ames, Iowa 50011
January 22, 2010

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), 3143 Compile-Time Error Detection (CTED) tests and a CTED evaluation tool have been developed. The CTED evaluation tool 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. All tests, the CTED evaluation tool and test results for the Berkeley, Cray, GNU and HP UPC compilers are freely available at http://hpcgroup.public.iastate.edu/CTED/.

Keywords: UPC, compile-time error detection

1. Introduction

Unified Parallel C is an extension of C for parallel execution on shared or distributed memory parallel machines [1], [2]. 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 [4]. Providing a productive programming environment for UPC will encourage new scientific applications to be written in UPC instead of MPI [3]. 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. Having compilers and run-time systems detect errors and issue high-quality error messages can greatly increase programmer productivity when developing and maintaining application programs by reducing debugging time. Some errors cannot be detected until run-time since the information needed to detect these errors is not available at compile-time.

Application programs are usually developed by (a) first 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 validate the program using problems for which answers are known. Usually programmer productivity is increased when program errors are detected at compile-time errors rather than at run-time.

Over two thousand **Run-Time** Error Detection (RTED) tests for UPC along with a RTED evaluation tool have been developed by Iowa State University's (ISU's) High Performance Computing (HPC) Group. A description of this work along with evaluation results for the Cray,

Berkeley, HP and GNU UPC compilers can be found in [9]. The focus of this paper is ISU's HPC Group's work for detecting UPC **compile-time** errors.

With funding from the US Department of Defense and from the Extreme Scale System Center at Oak Ridge National Laboratory 3143 UPC compile-time error detection (CTED) tests have been developed by ISU's HPC Group for evaluating parallel error detection capabilities of UPC compilers. Each test contains exactly one UPC compile-time error. A CTED evaluation tool has been developed for compiling these tests, assigning a score from 0 to 5 based on the quality of the error message, calculating the averages of these scores for each error category and reporting results.

These tests and the CTED evaluation tool 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. In addition, vendors could use the CTED tests, recommended error messages and the CTED evaluation tool to evaluate and improve the compile-time error detection capabilities of their UPC compilers.

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

2. Background

Unified Parallel C (UPC) is an extension of the ISO C 99 programming language designed for shared and distributed memory high performance parallel machines. UPC uses a single shared, partitioned address space, where variables may be directly read and written by any processor, but each variable is physically associated with a single processor [4].

The UPC Compiler Group at the University of California Berkeley/Lawrence Berkeley National Laboratory has developed the Berkeley UPC compiler. The goal of the Berkeley UPC compiler group is to develop an open-source, portable, high performance implementation of UPC for large-scale multiprocessors, PC clusters, and clusters of shared memory multiprocessors [4].

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 [5].

UPC work at Michigan Technological University 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 [6].

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 will facilitate users in identifying bottlenecks

in their programs and will serve as a testbed for advanced analysis techniques aimed at increasing programmer productivity [7].

When this work was done, Cray, Berkeley, HP and GNU had UPC compilers and a UPC compiler was under development by IBM. However, the GNU and HP UPC compilers did not support UPC IO.

3. Methodology

This section summarizes the methodology used to develop compile-time error tests and the CTED evaluation tool. 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. 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 “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. 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.

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 were assigned as follows:

- A score of 0 is given when the error was 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.

Different compilers may issue different messages (with different error names) for the same compile-time error. The CTED evaluation tool 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 are evaluated by the CTED evaluation tool as follows:

- For each test and score, a scoring script 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 evaluation.

4. Examples

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

Example 1: Applying a binary operator with incorrect operands.

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.

Example 2: Using an uninitialized pointer.

c_H_2_1 .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_1.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_l.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.

Example 3. An array declarator error when compiled with the dynamic threads environment option.

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];
     ^
```

Example 4. An out-of-bounds array access error.

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

5. Results

Table 1 in this section presents the scores when compiling the 3143 UPC CTED tests using the Berkeley, Cray, GNU and HP UPC compilers. Current results are listed on the web site http://hpcgroup.public.iastate.edu/CTED/.

UPC compile-time error category	Berkeley	Cray	GNU	HP
explicitly disallowed statements	2.84	2.74	2.96	2.58
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 functions calls	0.00	0.00	0.00	0.00
argument errors in UPC functions	0.05	0.07	0.06	0.09
wrong order of UPC statements and function calls	0.00	0.44	0.00	0.08
uninitialized variables	0.00	1.97	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	0.16	0.07	0.00	0.09
undefined UPC operations	0.00	0.21	0.16	0.21
warnings	0.00	0.06	0.00	1.84
average of the above scores	0.25	0.55	0.27	0.73

Table 1. UPC CTED test results for each error category.

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. The GNU and HP UPC compilers do not support UPC IO so they scored zero on these tests. The following summarizes the results:

- all the compilers achieved a score of nearly 3.0 in the “explicitly disallowed statements” category
- the Cray and HP UPC compilers scored about two in the "uninitialized variables" category
- the HP UPC compiler scored about two in the "warnings" category
- the scores for all other categories and vendors were low.

6. Conclusions

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), 3143 compile-time error tests and a CTED evaluation tool have been developed. For each error message issued, the CTED evaluation tool 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. The CTED evaluation tool calculates averages over each error category and then prints the results. All tests and the CTED evaluation tool are freely available at http://hpcgroup.public.iastate.edu/CTED/.

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 except for the error category “explicitly disallowed statements”.

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.

7. References

1. T. El-Ghazawi, W. Carlson, T. Sterling and K. Yelick; *UPC: Distributed Shared Memory Programming*, Wiley-Interscience, 2005.
2. Unified Parallel C (Wikipedia): http://en.wikipedia.org/wiki/Unified_Parallel_C
3. M. Snir, S Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI-The Complete Reference,* The MIT Press, 1998.
4. The Berkeley Unified Parallel C: http://upc.lbl.gov/
5. Unified Parallel C at George Washington University: http://upc.gwu.edu/
6. UPC Projects at Michigan Technological University: http://www.upc.mtu.edu/
7. High Performance Computing and Simulation Laboratory, University of Florida: http://www.hcs.ufl.edu/upc/
8. High Performance Computing Group, Iowa State University: http://hpcgroup.public.iastate.edu

9. Glenn R. Luecke, James Coyle, James Hoekstra, Marina Kraeva, Ying Xu, Elizabeth Kleiman, Olga Weiss; *Evaluating Error Detection Capabilities of UPC Run-time Systems*, The 3rd Conference on Partitioned Global Address Space (PGAS) Programming Models, October 2009, The George Washington University, Virginia, USA.