CIRM - Dynamic Error Detection

Peter Pirkelbauer
Center for Applied Scientific Computing (CASC)
Lawrence Livermore National Laboratory

This work was funded by the Department of Defense and used elements at the Extreme Scale Systems Center, at Oak Ridge. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94551

UCRL-LLNL-PRES-504931
Overview

1 Code Instrumentation and Runtime Monitor
   - Motivation
   - Sequential Codes
   - Parallel Codes
   - Results
   - Summary and Future Work
Example

Return Invalid Pointer

```c
int* foo() {
    int res = ...;
    return &res;
}
```

storage goes out of scope
res becomes invalid
Motivation

- Cost of software bugs is significant estimated at 0.6% of GDP [National Institute of Standards & Technology, 2002]
- Bug Detection Tools
  Valgrind, Insure++, Purify, ...
- Error Detection Benchmarks Suites
  RTED [Luecke et al., 2009b]
Bug Detection Tools

Dynamic Error Detection

Valgrind

binary code

static

dynamic

Insure++
Purify

CIRM

Coverity

CCured

Klocwork
Sequential Runtime Error Categories

Detected Runtime Errors

- C-style errors
  - out-of-bounds accesses, uninitialized variables, tangling pointers, arithmetic overflow/underflow
- C-library functions
  - arguments violate precondition
- Mismatches in memory allocation and deallocation methods
ROSE Overview

www.roseCompiler.org

ROSE Compiler Infrastructure

C/UPC/C++ Fortran/OpenMP Source Code
Software Binary Code

EDG Front-end/ Open Fortran Parser
Binary Disassembler

Analyzes Transformations Optimizations

ROSE-based Tools

IR (AST)

Unparser

2009 Winner

www.roseCompiler.org
ROSE-CIRM Architecture (Sequential)

**Code Instrumentation**
- Source Code
- ROSE compiler Infrastructure (C, C++, UPC)

**Compilation**
- Monitored Source Code
- Compiler (gcc, g++, upc, ...)
- Monitored Binary Code

**Monitored Execution**
- Unsafe operation System State updates
- Validation
- Runtime Monitor

Test programs

RTED benchmark suite
Original Code

```c
int* foo() {

    int res = ...;

    return &res;
}
```
int* foo() {

    int res = ...;
    cirmCreateVar(&res, "int", cirmInitialized);

    return &res;
}

text: creates variable record
```c
int* foo() {
    cirmScopeGuard guard;
    int res = ...;
    cirmCreateVar(&res, "int", cirmInitialized);
    int* ptr = &res;
    return ptr;
}
```
Instrumented Code

```c
int* wrapped_foo() {
    cirmScopeGuard guard;
    int res = ...;
    cirmCreateVar(&res, "int", cirmInitialized);
    int* ptr = &res;

    return ptr;
}

int* foo() {
    int* res = wrapped_foo();
    cirmValidatePtr(res);
    return res;
}
```
Unified Parallel C (UPC)

UPC extends C99

- Partitioned global address space (PGAS)
- Language constructs for parallelism
  - shared pointers, parallel for loop, memory consistency model

Stack

Heap

UPC Shared Memory

shared access

exclusive access
Parallel Runtime Error Categories

Detected Runtime Errors

- C-style error in the UPC shared space
  out-of-bounds accesses, uninitialized variables, tangling pointers, arithmetic overflow/underflow

Not Yet Implemented

- Parallelism related errors
  deadlocks, livelocks, race conditions
- UPC-library functions arguments violate precondition
shared[] int *values = upc_all_alloc(...);
cirmCreateHeap(values, ...);
cirmInitVariable(&values);
cirmAccessArray(&values[MYTHREAD], &values[0]); // bounds check
values[MYTHREAD] = ...;
cirmInitVar(&values[MYTHREAD], ...);
CIRM Runtime System (Parallel)

Instrumented Code

```c
shared[] int *values = upc_all_alloc(...);
cirmCreateHeap(values, ...);
cirmInitVariable(&values);

// bounds check
cirmAccessArray(&values[MYTHREAD], &values[0]);
values[MYTHREAD] = ...;
cirmInitVar(&values[MYTHREAD], ...);
```
Runtime Monitor Coordination - Concurrent Access (1)

Instrumented Code

```c
// shared int val;
if (MYTHREAD == 0) {
    val = compute(...);
    cirmInitVariable(&val, ...);
}
cirmEnterBarrier();
upc_barrier;
cirmExitBarrier();
cirmAccessVar(&val, ...);
printf("%d\n", val);
```

Update messages are processed after a barrier.
Instrumented Code

```c
// shared int val;
if (MYTHREAD == 0) {
    val = compute(...);
    cirmInitVariable(&val, ...);
}
```

Under a race CIRM may report a spurious error.
(the check will never spuriously succeed).

```c
cirmAccessVar(&val, ...);
printf("%d\n", val);
```
Runtime Monitor Coordination - Early Release

Instrumented Code

```c
shared[] int *values = upc_all_alloc(...);
values[idx] = compute(idx);

// upc_barrier;
if (MYTHREAD == 0) {
    upc_free(ptr);
}
```

- **missing barrier**
- **Race can lead to early release**

Peter Pirkelbauer (LLNL)
Dynamic Error Detection
Instrumented Code

```c
shared[] int *values = upc_all Alloc(...);
cirmArrayAccess(&values[0] &values[0];
values[idx] = compute(idx);
cirmInitVariable(&values[...], ...);
// upc_barrier;
if (MYTHREAD == 0) {
    cirmEnterHeapUpdate();
    cirmFreeMem(&ptr);
    upc_free(ptr);
    cirmExitHeapUpdate();
}
```

- **missing barrier**
- **isolate destructive updates**
- **Race can lead to early release**
Address Abstraction

Implemented for GCCUPC [Funck, 2006]

UPC Thread 0

Stack

UPC Shared Memory

Heap

char* p

shared char* p_shared

UPC Thread 1

Stack

UPC Shared Memory

Heap
Luecke et al.: RTED Benchmark Suite for C++03 [Luecke et al., 2009b]

<table>
<thead>
<tr>
<th>Category</th>
<th>Number of tests</th>
<th>Correctly Identified (in percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation deallocation errors</td>
<td>109</td>
<td>104 (95%)</td>
</tr>
<tr>
<td>Array index out of bound</td>
<td>332</td>
<td>329 (99%)</td>
</tr>
<tr>
<td>Floating point errors</td>
<td>17</td>
<td>17 (100%)</td>
</tr>
<tr>
<td>Input output errors</td>
<td>28</td>
<td>18 (64%)</td>
</tr>
<tr>
<td>Memory leaks</td>
<td>42</td>
<td>38 (90%)</td>
</tr>
<tr>
<td>Pointer errors</td>
<td>157</td>
<td>155 (99%)</td>
</tr>
<tr>
<td>String errors</td>
<td>40</td>
<td>40 (100%)</td>
</tr>
<tr>
<td>Uninitialized variables</td>
<td>221</td>
<td>213 (96%)</td>
</tr>
</tbody>
</table>
Luecke et al.: RTED Benchmark Suite for UPC [Luecke et al., 2009a]

<table>
<thead>
<tr>
<th>Category</th>
<th>Number of tests</th>
<th>Correctly Identified (in percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out of bounds accesses (indices)</td>
<td>726</td>
<td>685 (94%)</td>
</tr>
<tr>
<td>Out of bounds accesses (pointers)</td>
<td>160</td>
<td>150 (94%)</td>
</tr>
<tr>
<td>Uninitialized memory reads</td>
<td>64</td>
<td>62 (97%)</td>
</tr>
<tr>
<td>Dynamic memory handling related</td>
<td>10</td>
<td>10 (100%)</td>
</tr>
</tbody>
</table>
Tests: Performance

El-Ghazawi et al.: Distributed Shared Memory Programming [El-Ghazawi et al., 2003]

- 80 elements per dimension
- 8 Threads
- Intel X5680 6x2 cores @ 3.3Ghz
- 24GByte Memory
- Red Hat Linux 5.6
- gccupc 4.5.1.2, g++ 4.1.2
### Static Analysis Comes to Rescue

- **Reaching definition**
  - eliminates local initialization checks

- **Local escape analysis**
  - eliminates variable tracking

- **Interval analysis**
  - eliminates local bounds checks

- ...  

### Integrate Checking into Instrumented Code

Implemented arithmetic overflow/underflow checks
- → performance overhead is 20%
Summary and Future Work

- Integrate static analysis to improve sequential checks
- Develop static analysis to accelerate checking parallel codes
  - absence of race conditions in certain code segments
  - to use less expensive checking mechanisms
  - reduce communication overhead

1 Runtime Detection of C-Style Errors in UPC Code. [Pirkelbauer et al., 2011]
Thank You!

*UPC: Distributed Shared-Memory Programming.*

Gary Funck.

GPC/UPC 4.0, "flexible heap" design overview.

Glenn R. Luecke, James Coyle, James Hoekstra, Marina Kraeva, Ying Xu, Elizabeth Kleiman, and Olga Weiss.

Evaluating error detection capabilities of UPC run-time systems.

Glenn R. Luecke, James Coyle, James Hoekstra, Marina Kraeva, Ying Xu, Mi-Young Park, Elizabeth Kleiman, Olga Weiss, Andrew Wehe, and Melissa Yahya.
The importance of run-time error detection.
In *Third Parallel Tools Workshop*, 2009.

National Institute of Standards & Technology.

*The Economic Impacts of Inadequate Infrastructure for Software Testing.*
RTI (Health, Social, and Economics Research), May 2002.

Peter Pirkelbauer, Chunhua Liao, Thomas Panas, and Daniel Quinlan.

Runtime detection of C-style errors in UPC code.