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

```
int∗ foo() {
  int res = ...;

  return &res;
}
```

storage goes out of scope
res becomes invalid

Runtime Error Detection for C, C++03 and UPC

Motivation

- Cost of software bugs is significant

  estimated at 0.6% of GDP [National Institute of Standards & Technology, 2002]

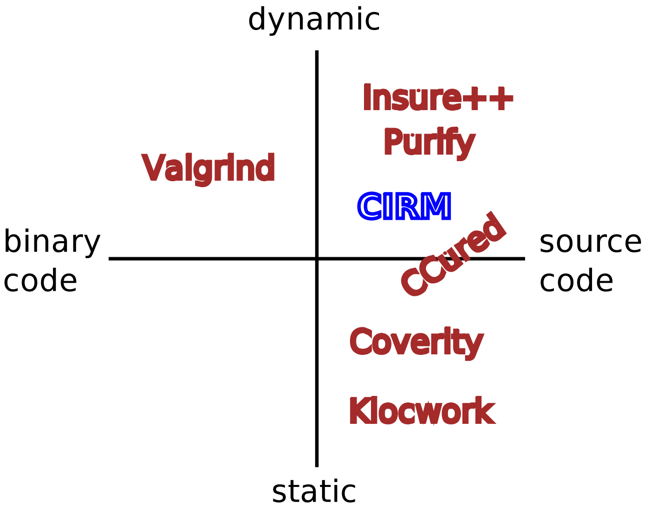
- Bug Detection Tools

  Valgrind, Insure++, Purify, ...

- Error Detection Benchmarks Suites

  RTED [Luecke *et al.*, 2009b]

Bug Detection Tools



dynamic

static

binary code

source code

Insure++

Purify

Valgrind

CIRM

CCured

Coverity

Klocwork

Sequential Runtime Error Categories

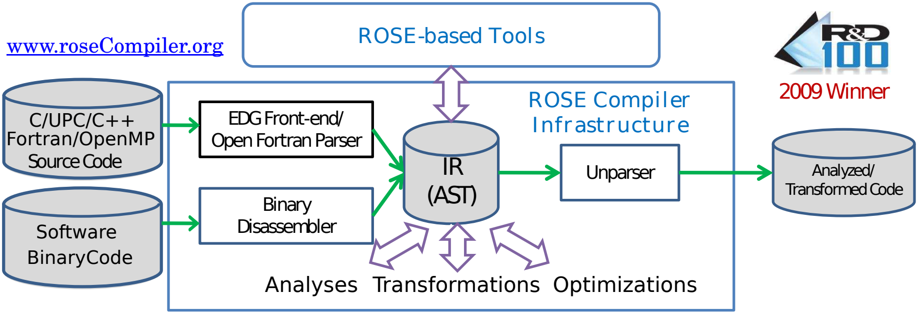
Detected Runtime Errors

- C-style errors

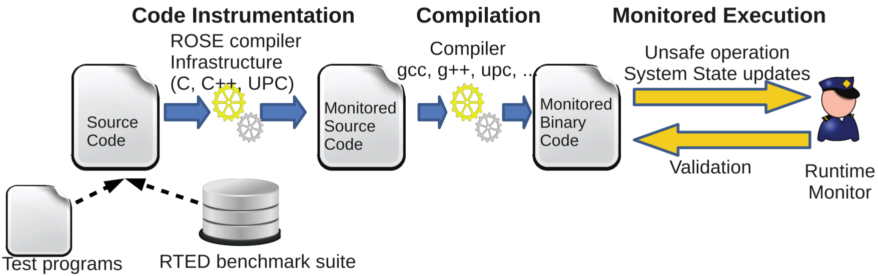
  out-of-bounds accesses, uninitialized variables, tangling pointers, arithmetic overflow/underflow
- C-library functions

  arguments violate precondition
- Mismatches in memory allocation and deallocation methods

ROSE Overview



ROSE-CIRM Architecture (Sequential)



Code Instrumentation: Scope and Pointer Tracking

Original Code

```
int* foo() {

   int res = ...;


   return &res;
}
```

Code Instrumentation: Scope and Pointer Tracking

Instrumented Code

```
int∗ foo() {

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

  return &res;
}
```

creates variable record

Code Instrumentation: Scope and Pointer Tracking

Instrumented Code

```
int∗ foo() {
  cirmScopeGuard guard;
  int res = ...;
  cirmCreateVar(&res, "int", cirmInitialized);
  int∗ ptr = &res;

  return ptr;
}
```

creates scope and local memory

temporarily stores result

RAII cleans-up stack and variables

Code Instrumentation: Scope and Pointer Tracking

Instrumented Code

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

   return ptr;
}
```

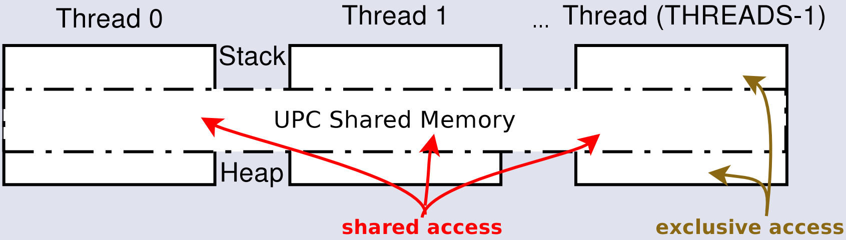
```
int* foo() {
   int* res = wrapped_foo();
   cirmValidatePtr(res);
   return res;
}
```

validate return value

Unified Parallel C (UPC)

UPC extends C99

- Partitioned global address space (PGAS)



- Language constructs for parallelism
  shared pointers, parallel for loop, memory consistency model

Parallel Runtime Error Categories

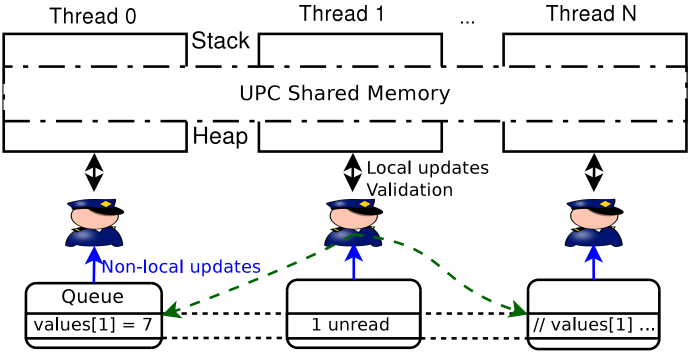
Detected Runtime Errors

- 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

CIRM Runtime System (Parallel)

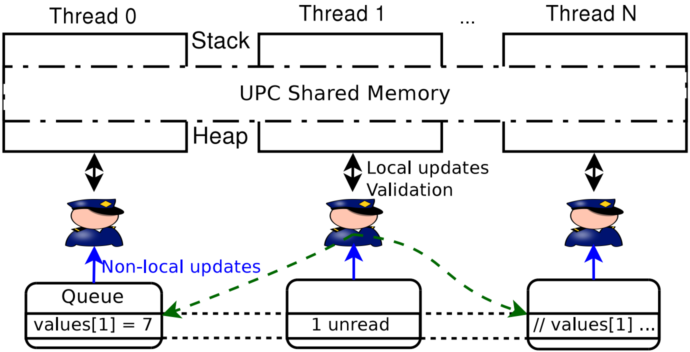


Instrumented Code

```
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

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

cirmAccessArray(&values[MYTHREAD], &values[0]); // bounds check
values[MYTHREAD] = ...;
cirmInitVar(&values[MYTHREAD], ...);
```

Runtime Monitor Coordination - Concurrent Access (1)

Instrumented Code

```
// 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.

Runtime Monitor Coordination - Concurrent Access (2)

Instrumented Code

```
// 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).

```
cirmAccessVar(&val, ...);
printf("%d\n", val);
```

Runtime Monitor Coordination - Early Release

Instrumented Code

```
shared[] int ∗values = upc_all_alloc(...);

values[idx] = compute(idx);

// upc_barrier;
```

missing barrier

```
if (MYTHREAD == 0) {

   upc_free(ptr);
```

Race can lead to early release

```
}
```

Runtime Monitor Coordination - Early Release

Instrumented Code

```
shared[] int ∗values = upc_all_alloc(...);

cirmArrayAccess(&values[0] &values[idx]);
values[idx] = compute(idx);
cirmInitVariable(&values[...], ...);

// upc_barrier;

if (MYTHREAD == 0) {
  cirmEnterHeapUpdate();
  cirmFreeMem(&ptr);
  upc_free(ptr);
  cirmExitHeapUpdate();
}
```

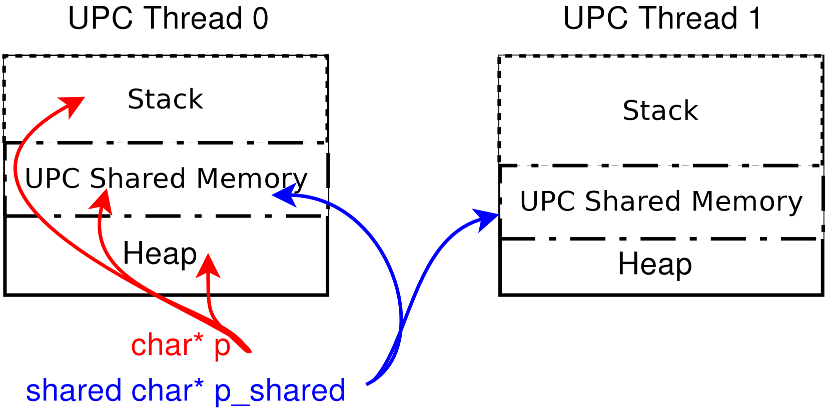
missing barrier

isolate destructive updates

Race can lead to early release

Address Abstraction

Implemented for GCCUPC [Funck, 2006]



Tests: Error Detection Benchmark (C++03)

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

Category	Number of tests	Correctly Identified (in percent)	
Allocation deallocation errors	109	104	(95%)
Array index out of bound	332	329	(99%)
Floating point errors	17	17	(100%)
Input output errors	28	18	(64%)
Memory leaks	42	38	(90%)
Pointer errors	157	155	(99%)
String errors	40	40	(100%)
Uninitialized variables	221	213	(96%)

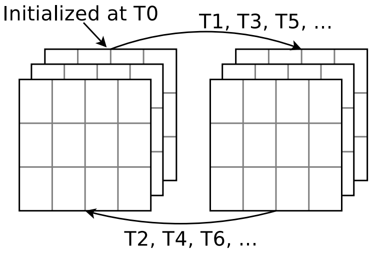
Tests: Error Detection Benchmark (UPC)

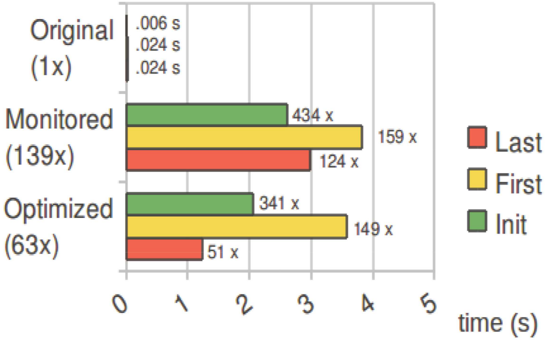
Luecke et al.: RTED Benchmark Suite for UPC [Luecke *et al.*, 2009a]

Category	**Number of tests**	**Correctly Identified (in percent)**	
Out of bounds accesses (indices)	726	685	(94%)
Out of bounds accesses (pointers)	160	150	(94%)
Uninitialized memory reads	64	62	(97%)
Dynamic memory handling related	10	10	(100%)

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

Improving Performance

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!

Tarek El-Ghazawi, William Carlson, Thomas Sterling, and Katherine Yelick.
UPC: Distributed Shared-Memory Programming.
Wiley-Interscience, 2003.

Gary Funck.
GPC/UPC 4.0, "flexible heap" design overview.
Technical report, Intrepid Technology Inc., Sep 2006.

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.
In *Third Conference on Partitioned Global Address Space Programing Models*, PGAS '09, pages 7.1 – 7.4, New York, NY, USA, 2009. ACM.

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.
In *5th Conference on Partitioned Global Address Space Models (PGAS)*, 2011.