Call Paths for Pin Tools

Milind Chabbi, Xu Liu, and John Mellor-Crummey
Department of Computer Science
Rice University

Comp 600, January 27, 2014
Star Graduate Student
Star Graduate Student
Invention!

Data race detection tool

Inventor
Found Client for His Tool

Data race detection tool

Inventor

User
Found Client for His Tool

Data race detection tool

User
Client Uses the Tool

Data race detection tool

User
Client Uses the Tool

Data race detection tool

User
Client Uses the Tool

Data race detection tool

User

> tool ./myApp
Client Uses the Tool

Data race detection tool

>tool ./myApp

Data race detected!

User
Where is My Concurrency Bug?

Data race detection tool

- 100s of files, 1000s of LOC, numerous functions
- Existing tools lack this capability
- We demonstrate how it is possible with acceptable overhead
Where is My Concurrency Bug?

Data race detection tool

- 100s of files, 1000s of LOC, numerous functions
- Existing tools lack this capability
- We demonstrate how it is possible with acceptable overhead
Need Better Diagnostic Capabilities for Tools

Data race detection tool

Thread 1

Thread 2

User

```
Foo() {
  *ptr = 100;
}

Bar() {
  x = *ptr;
}
```
Need Better Diagnostic Capabilities for Tools

Data race detection tool

Thread 1
- M()
- N()
- B()

Bar() {
  x = *ptr;
}

Thread 2
- P()
- Q()
- D()

Foo() {
  *ptr = 100;
}

User
Need Better Diagnostic Capabilities for Tools

Data race detection tool

Thread 1

Main()

A()

B()

Thread 2

MyTask()

P()

Q()

User

Calling context enhances tool’s capability/usability

Foo() {
   *ptr = 100;
}

Bar() {
   x = *ptr;
}
How Data Race Detection Works

- Tool executes the program
- Tool monitors every memory access by each thread
- Tool maintains abbreviated history of previous accesses (thread id) for each memory address
- Tool inspects the access history and determines if any conflicting accesses happen in parallel
Challenges of Providing Calling Context

- **Unwinding** can collect current calling context
- Calling context of previous accesses is lost

Thread 1
- `Main()`
  - `A()`
  - `B()`

Thread 2
- `MyTask()`
  - `P()`
  - `Q()`

```
Foo() {
  *ptr = 100;
}
```
```
Bar() {
  x = *ptr;
}
```
Challenges of Providing Calling Context

- **Unwinding** can collect current calling context
- Calling context of previous accesses is lost

```
Main()
  A()
  B()
  Bar() { x = *ptr; }

Foo() {
  *ptr = 100; }
```

```
MyTask()
  P()
  Q()
```

```
Bar() {
  x = *ptr; }
```

```
Foo() {
  *ptr = 100; }
```
Naive Solution: Maintain a History of Contexts

Unwind and store call path on each access

Thread 1

Main()
A()
B()

Bar() { x = *ptr; }

Thread N

MyTask()

W()
X()
Y()

Z() {...
... = *ptr; }

P()
Q()

Foo() { *ptr = 100; }
Naive Solution: Maintain a History of Contexts

Unwind and store call path on each access

Thread 1

Main() → A() → B() → Bar() { x = *ptr; }

Thread N

MyTask() → W() → X() → Y() → Z() { ... = *ptr; } → Foo() { *ptr = 100; }

...
Naive Solution: Maintain a History of Contexts

Unwind and store call path on each access

Thread 1

Main()

A()

B()

Bar() {
  x = *ptr;
}

Thread N

W()

X()

Y()

Z() {
  ... = *ptr;
}

MyTask()

P()

Q()

Foo() {
  *ptr = 100;
}
Naive Solution: Maintain a History of Contexts

Unwind and store call path on each access

Thread 1

Main()
  \rightarrow
  A()
  \rightarrow
  B()

Bar() { 
  x = *ptr;
}

Thread N

MyTask()
  \rightarrow
  P()

MyTask()
  \rightarrow
  Q()

W()

X()

Y()

Z() { 
  ... = *ptr;
}

Foo() { 
  *ptr = 100;
}
Overheads of Naive Solution

**Naive Solution**
Unwind and record the calling context on each memory access

1. **Space overhead** for maintaining many calling contexts
2. **Time overhead** for call stack unwinding at each memory access
Frameworks for Fine-Grained Program Monitoring

“Getting calling contexts using VG_(get_StackTrace) is done via stack unwinding. Unwinding for each memory access will be very slow.”

“It will slow down execution by a factor of several thousand compared to native execution -- I'd guess -- so you'll wind up with something that is unusably slow on anything except the smallest problems.”

“If you tried to invoke Thread::getCallStack on every memory access there would be very serious performance problems … your program would probably never reach main.”

- No support for collecting calling contexts
- We built it ourselves—CCTLib
- Demonstrate how it is possible to gather calling context ubiquitously with CCTLib
Many Tools Require Fine-grained Program Monitoring

- **Performance analysis tools**
  - Cache simulators
  - Reuse-distance analysis
  - False sharing detection
  - Memory / computation redundancy

- **Software correctness**
  - Taint analysis
  - Malware detection
  - Memory leak / array out of bounds

- **Many other tools, e.g.,**
  - Debugging, testing, resiliency, replay, etc.
Store History of Contexts Compactly

Space bloat problem

A
B
D
A
B
E
A
C
F
A
G

0x00
0x01
0x02
0x03

shadow memory
Store History of Contexts Compactly

Space bloat problem

Solution
• Call paths share common prefix
• Store call paths as a calling context tree (CCT)
• One CCT per thread
Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

```
Main()
   P()
   Q()
   Foo() { 
      *ptr = 100;
      x = 42;
   }
```
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

```c
Foo() {
    *ptr = 100;
    x = 42;
}
```

Diagram:
```
Main() -> P() -> Q() -> Foo()
```
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Main()
P()
Q()

Foo() {
    *ptr = 100;
    x = 42;
}
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```c
Foo() {
    *ptr = 100;
    x = 42;
}
```
Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```c
Main() {
    P();
    Q();
    Foo() {
        *ptr = 100;
        x = 42;
    }
}
```
Shadow Stack to Avoid Unwinding Overhead

Problem:
Unwinding overhead

Solution:
Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

Main()

Foo() {
*ptr = 100;
x = 42;
}
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```c
Foo() {
    *ptr = 100;
    x = 42;
}
```
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```c
main()
    P();
    Q();

Foo() {
    *ptr = 100;
    x = 42;
}
```
Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```
Foo() {
    *ptr = 100;
    x = 42;
}
```
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Main()
P()
Q()
Foo() {
    *ptr = 100;
    x = 42;
}

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

Main()
P()
Q()
Foo() {
    *ptr = 100;
    x = 42;
}

call
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```
Main()
P()
Q()
Foo() {
    *ptr = 100;
    x = 42;
}
*ptr = 100;
x = 42; }
```

```
Main()
P()
Q()
Foo() {
    *ptr = 100;
    x = 42;
}
*ptr = 100;
x = 42; }
```
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```
Main()
P()
Q()
Foo() {
    *ptr = 100;
    x = 42;
}
R()
```

```
Main()
P()
Q()
Foo() {
    *ptr = 100;
    x = 42;
}
R()
```
Shadow Stack to Avoid Unwinding Overhead

Problem: Unwinding overhead

Solution: Reverse the process. Eagerly build a replica/shadow stack on-the-fly.

```
Main()
P()
Q()

Foo() {
    *ptr = 100;
    x = 42;
}

*ptr = 100;
x = 42;
```

Tools can obtain pointer to the current context via “CTX” in constant time.
CTXT Update Cost

Main()

P()

Q()

W() ... Z()
CTXT Update Cost

Main()
P()
Q()
W()
Z()
CTXT Update Cost

Return to caller is constant time operation
Callee Lookup Could be Costly

\[
\text{Main()}
\]

\[
\text{P()}
\]

\[
\text{Q()}
\]

\[
\text{W() \quad \ldots \quad Z()}
\]

\[
\text{CTX}
\]
Callee Lookup Could be Costly

Each “Call” from caller “Q” to its callees incurs a lookup cost
Callee Lookup Could be Costly

Each “Call” from caller “Q” to its callees incurs a lookup cost
Accelerating Lookup Cost with Splay Trees

Splay tree [“Self-adjusting binary search trees” by Sleator et al. 1985] ensures frequently called functions are near the root of the tree.
Accelerating Lookup Cost with Splay Trees

Splay tree [“Self-adjusting binary search trees” by Sleator et al. 1985] ensures frequently called functions are near the root of the tree.
Accelerating Lookup Cost with Splay Trees

Splay tree [“Self-adjusting binary search trees” by Sleator et al. 1985] ensures frequently called functions are near the root of the tree.
Accelerating Lookup Cost with Splay Trees

Splay tree [“Self-adjusting binary search trees” by Sleator et al. 1985] ensures frequently called functions are near the root of the tree.
Other Complications in Real Programs

- Attributing to instructions/source lines (not just functions)

```c
Main()
P()
Q()
Foo() {
    *ptr = 100;
x = 42;
}
R()
```
Other Complications in Real Programs

- Attributing to instructions/source lines (not just functions)

```c
Main()
P()
Q()
Foo() {
  *ptr = 100;
  x = 42;
}
R()
```

CTXT = Foo: line 1
Other Complications in Real Programs

- Attributing to instructions/source lines (not just functions)

```c
Main()
P()
Q()
Foo() {
    *ptr = 100;
x = 42;
}
R()
```

```
CTXT = Foo: line 1
CTXT = Foo: line 2
```
Other Complications in Real Programs

- Attributing to instructions/source lines (not just functions)
- Complex control flow
  - Signal handling
  - Setjmp-Longjmp
  - C++ exceptions (try-catch)
- Thread creation and destruction
  - Maintaining parent-child relationships between threads
  - Scalability to large number of threads
Data-Centric Attribution in CCTLib

- Associate each data access to the corresponding data object
- **Data object:**
  - Dynamic allocation ➔ Call path of allocation site
  - Static objects ➔ Variable name

```c
int * Create()
{
    return malloc(...);
}

void Update(int * ptr)
{
    for( ... )
        ptr[i]++;
}

Main()
{
    int * p = Create();
    Update(p);
}
```
Data-Centric Attribution in CCTLib

```c
int * Create()
{
    return malloc(...);
}

void Update(int * ptr) {
    for ( ... )
        ptr[i]++;
}

Main()
{
    int * p = Create();
    Update(p);
}
```

- Associate each data access to the corresponding data object
- Data object:
  - Dynamic allocation ➔ Call path of allocation site
  - Static objects ➔ Variable name
Data-Centric Attribution in CCTLib

```c
int * Create(){
    return malloc(...);
}
void Update(int * ptr) {
    for( ... )
        ptr[i]++;
}
Main(){
    int * p = Create();
    Update(p);
}
```

- Associate each data access to the corresponding data object
- Data object:
  - Dynamic allocation ➔ Call path of allocation site
  - Static objects ➔ Variable name
Details of Data-Centric Attribution

• How to perform data-centric attribution
  ✦ Record all `<AddressRange, VariableName>` tuples in a map
  ✦ Intercept all allocation/free routines and maintain `<AddressRange, CallPath>` tuples in a map
  ✦ On each memory access search these maps for the address

• Problems:
  ✦ Searching maps on each access is expensive
  ✦ Maps need to be concurrent for threaded programs
Data-Centric Attribution via Balanced Trees

- **Observation:**
  - Updates to maps are infrequent
  - Lookups in maps are frequent

- **Solution #1: sorted map**
  - Keep N objects and associated address range in balanced binary trees
    - Low memory cost—O(N), moderate lookup cost—O(log N)
    - Concurrent access is handled by a novel replicated tree data structure
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

• Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory
Data-Centric Attribution via Shadow Memory

- Solutions #2: shadow memory

- Have shadow memory for each memory cell and record a handle to the corresponding data object in the shadow memory
  - Low lookup cost—$O(1)$, high memory cost
  - Concurrent access is not a problem

- CCTLib supports both solutions, clients can choose
Evaluation

- Time overhead
- Memory overhead
- Scaling on multithreaded programs
- Real world, long running programs
## Time Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Time in sec</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution on each instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>276.26</td>
<td>14x</td>
<td>22x</td>
<td>28x</td>
</tr>
<tr>
<td>bzip2</td>
<td>111.71</td>
<td>19x</td>
<td>32x</td>
<td>42x</td>
</tr>
<tr>
<td>gcc</td>
<td>44.61</td>
<td>23x</td>
<td>35x</td>
<td>44x</td>
</tr>
<tr>
<td>h264ref</td>
<td>260.12</td>
<td>31x</td>
<td>48x</td>
<td>67x</td>
</tr>
<tr>
<td>hmer</td>
<td>326.32</td>
<td>21x</td>
<td>30x</td>
<td>47x</td>
</tr>
<tr>
<td>libquantum</td>
<td>462.38</td>
<td>22x</td>
<td>39x</td>
<td>46x</td>
</tr>
<tr>
<td>mcf</td>
<td>319.97</td>
<td>6x</td>
<td>10x</td>
<td>15x</td>
</tr>
<tr>
<td>omnetpp</td>
<td>352.30</td>
<td>14x</td>
<td>23x</td>
<td>34x</td>
</tr>
<tr>
<td>Xalan</td>
<td>294.80</td>
<td>32x</td>
<td>50x</td>
<td>65x</td>
</tr>
<tr>
<td>ROSE</td>
<td>23.64</td>
<td>30x</td>
<td>41x</td>
<td>49x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>99.28</td>
<td>17x</td>
<td>29x</td>
<td>40x</td>
</tr>
<tr>
<td>LULESH</td>
<td>67.29</td>
<td>20x</td>
<td>36x</td>
<td>48x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td><strong>19x</strong></td>
<td><strong>30x</strong></td>
<td><strong>41x</strong></td>
<td><strong>41x</strong></td>
</tr>
</tbody>
</table>
### Time Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Program Time in sec</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution on each instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>276.26</td>
<td>14x</td>
<td>22x</td>
<td>28x</td>
</tr>
<tr>
<td>bzip2</td>
<td>111.71</td>
<td>19x</td>
<td>32x</td>
<td>42x</td>
</tr>
<tr>
<td>gcc</td>
<td>44.61</td>
<td>23x</td>
<td>35x</td>
<td>44x</td>
</tr>
<tr>
<td>h264ref</td>
<td>260.12</td>
<td>31x</td>
<td>48x</td>
<td>67x</td>
</tr>
<tr>
<td>hmmer</td>
<td>326.32</td>
<td>21x</td>
<td>30x</td>
<td>47x</td>
</tr>
<tr>
<td>libquantum</td>
<td>462.38</td>
<td>22x</td>
<td>39x</td>
<td>46x</td>
</tr>
<tr>
<td>mcf</td>
<td>319.97</td>
<td>6x</td>
<td>10x</td>
<td>15x</td>
</tr>
<tr>
<td>omnetpp</td>
<td>352.30</td>
<td>14x</td>
<td>23x</td>
<td>34x</td>
</tr>
<tr>
<td>Xalan</td>
<td>294.80</td>
<td>32x</td>
<td>50x</td>
<td>65x</td>
</tr>
<tr>
<td>ROSE</td>
<td>23.64</td>
<td>30x</td>
<td>41x</td>
<td>49x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>99.28</td>
<td>17x</td>
<td>29x</td>
<td>40x</td>
</tr>
<tr>
<td>LULESH</td>
<td>67.29</td>
<td>20x</td>
<td>36x</td>
<td>48x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td><strong>19x</strong></td>
<td><strong>30x</strong></td>
<td><strong>41x</strong></td>
<td></td>
</tr>
</tbody>
</table>
# Time Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Time in sec</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution on each instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>276.26</td>
<td>14x</td>
<td>22x</td>
<td>28x</td>
</tr>
<tr>
<td>bzip2</td>
<td>111.71</td>
<td>19x</td>
<td>32x</td>
<td>42x</td>
</tr>
<tr>
<td>gcc</td>
<td>44.61</td>
<td>23x</td>
<td>35x</td>
<td>44x</td>
</tr>
<tr>
<td>h264ref</td>
<td>260.12</td>
<td>31x</td>
<td>48x</td>
<td>67x</td>
</tr>
<tr>
<td>hmmer</td>
<td></td>
<td></td>
<td></td>
<td>47x</td>
</tr>
<tr>
<td>libquantum</td>
<td></td>
<td></td>
<td></td>
<td>46x</td>
</tr>
<tr>
<td>mcf</td>
<td></td>
<td></td>
<td></td>
<td>15x</td>
</tr>
<tr>
<td>omnetpp</td>
<td>352.30</td>
<td>14x</td>
<td>23x</td>
<td>34x</td>
</tr>
<tr>
<td>Xalan</td>
<td>294.80</td>
<td>32x</td>
<td>50x</td>
<td>65x</td>
</tr>
<tr>
<td>ROSE</td>
<td>23.64</td>
<td>30x</td>
<td>41x</td>
<td>49x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>99.28</td>
<td>17x</td>
<td>29x</td>
<td>40x</td>
</tr>
<tr>
<td>LULESH</td>
<td>67.29</td>
<td>20x</td>
<td>36x</td>
<td>48x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td></td>
<td><strong>19x</strong></td>
<td><strong>30x</strong></td>
<td><strong>41x</strong></td>
</tr>
</tbody>
</table>

Source-to-source compiler from LLNL

3M LOC compiling 70K LOC

Deep call chains
## Time Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Time in sec</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution on each instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>276.26</td>
<td>14x</td>
<td>22x</td>
<td>28x</td>
</tr>
<tr>
<td>bzip2</td>
<td>111.71</td>
<td>19x</td>
<td>32x</td>
<td>42x</td>
</tr>
<tr>
<td>gcc</td>
<td>44.61</td>
<td>23x</td>
<td>35x</td>
<td>44x</td>
</tr>
<tr>
<td>h264ref</td>
<td>260.12</td>
<td>31x</td>
<td>48x</td>
<td>67x</td>
</tr>
<tr>
<td>hmmer</td>
<td></td>
<td></td>
<td></td>
<td>47x</td>
</tr>
<tr>
<td>libquantum</td>
<td></td>
<td></td>
<td></td>
<td>46x</td>
</tr>
<tr>
<td>mcf</td>
<td></td>
<td></td>
<td></td>
<td>15x</td>
</tr>
<tr>
<td>omnetpp</td>
<td></td>
<td></td>
<td></td>
<td>34x</td>
</tr>
<tr>
<td>Xalan</td>
<td>294.80</td>
<td>32x</td>
<td>50x</td>
<td>65x</td>
</tr>
<tr>
<td>ROSE</td>
<td>23.64</td>
<td>30x</td>
<td>41x</td>
<td>49x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>99.28</td>
<td>17x</td>
<td>29x</td>
<td>40x</td>
</tr>
<tr>
<td>LULESH</td>
<td>67.29</td>
<td>20x</td>
<td>36x</td>
<td>48x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td><strong>19x</strong></td>
<td><strong>30x</strong></td>
<td><strong>41x</strong></td>
<td><strong>41x</strong></td>
</tr>
</tbody>
</table>
## Time Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Time in sec</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution on each instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>276.26</td>
<td>14x</td>
<td>22x</td>
<td>28x</td>
</tr>
<tr>
<td>bzip2</td>
<td>111.71</td>
<td>19x</td>
<td>32x</td>
<td>42x</td>
</tr>
<tr>
<td>gcc</td>
<td>44.61</td>
<td>23x</td>
<td>35x</td>
<td>44x</td>
</tr>
<tr>
<td>h264ref</td>
<td>260.12</td>
<td>31x</td>
<td>48x</td>
<td>67x</td>
</tr>
<tr>
<td>hmmer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libquantum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>omnetpp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xalan</td>
<td>294.80</td>
<td>32x</td>
<td>50x</td>
<td>65x</td>
</tr>
<tr>
<td>ROSE</td>
<td>23.64</td>
<td>30x</td>
<td>41x</td>
<td>49x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>99.28</td>
<td>17x</td>
<td>29x</td>
<td>40x</td>
</tr>
<tr>
<td>LULESH</td>
<td>67.29</td>
<td>20x</td>
<td>36x</td>
<td>48x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td><strong>19x</strong></td>
<td><strong>30x</strong></td>
<td><strong>41x</strong></td>
<td></td>
</tr>
</tbody>
</table>

Hydrodynamics mini-app from LLNL
- Frequent data allocation and de-allocations
- Memory bounded
- Multithreaded, Poor scaling
## Time Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Time in sec</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution on each instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>276.26</td>
<td>14x</td>
<td>22x</td>
<td>28x</td>
</tr>
<tr>
<td>bzip2</td>
<td>111.71</td>
<td>19x</td>
<td>32x</td>
<td>42x</td>
</tr>
<tr>
<td>gcc</td>
<td>44.61</td>
<td>23x</td>
<td>35x</td>
<td>44x</td>
</tr>
<tr>
<td>h264ref</td>
<td>260.12</td>
<td>31x</td>
<td>48x</td>
<td>67x</td>
</tr>
<tr>
<td>hmmmer</td>
<td>326.32</td>
<td>21x</td>
<td>30x</td>
<td>47x</td>
</tr>
<tr>
<td>libquantum</td>
<td>462.38</td>
<td>22x</td>
<td>39x</td>
<td>46x</td>
</tr>
<tr>
<td>mcf</td>
<td>319.97</td>
<td>6x</td>
<td>10x</td>
<td>15x</td>
</tr>
<tr>
<td>omnetpp</td>
<td>352.30</td>
<td>14x</td>
<td>23x</td>
<td>34x</td>
</tr>
<tr>
<td>Xalan</td>
<td>294.80</td>
<td>32x</td>
<td>50x</td>
<td>65x</td>
</tr>
<tr>
<td>ROSE</td>
<td>23.64</td>
<td>30x</td>
<td>41x</td>
<td>49x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>99.28</td>
<td>17x</td>
<td>29x</td>
<td>40x</td>
</tr>
<tr>
<td>LULESH</td>
<td>67.29</td>
<td>20x</td>
<td>36x</td>
<td>48x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td><strong>19x</strong></td>
<td><strong>30x</strong></td>
<td><strong>41x</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Memory Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Original resident memory in MB</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution via binary tree</th>
<th>Data-centric attribution via shadow memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>230</td>
<td>1.16x</td>
<td>1.17x</td>
<td>1.34x</td>
<td>8.65x</td>
</tr>
<tr>
<td>bzip2</td>
<td>561</td>
<td>1.11x</td>
<td>1.12x</td>
<td>1.12x</td>
<td>8.03x</td>
</tr>
<tr>
<td>gcc</td>
<td>453</td>
<td>15.62x</td>
<td>25.97x</td>
<td>26.03x</td>
<td>36.38x</td>
</tr>
<tr>
<td>h264ref</td>
<td>37</td>
<td>2.49x</td>
<td>2.69x</td>
<td>2.91x</td>
<td>11.47x</td>
</tr>
<tr>
<td>hmmmer</td>
<td>15</td>
<td>4.38x</td>
<td>4.36x</td>
<td>5.13x</td>
<td>29.39x</td>
</tr>
<tr>
<td>libquantum</td>
<td>96</td>
<td>1.28x</td>
<td>1.30x</td>
<td>1.32x</td>
<td>11.91x</td>
</tr>
<tr>
<td>mcf</td>
<td>1677</td>
<td>1.02x</td>
<td>1.03x</td>
<td>1.03x</td>
<td>6.53x</td>
</tr>
<tr>
<td>omnetpp</td>
<td>170</td>
<td>1.87x</td>
<td>2.35x</td>
<td>3.76x</td>
<td>10.54x</td>
</tr>
<tr>
<td>Xalan</td>
<td>419</td>
<td>25.86x</td>
<td>38.60x</td>
<td>39.12x</td>
<td>46.63x</td>
</tr>
<tr>
<td>ROSE</td>
<td>380</td>
<td>64.38x</td>
<td>98.15x</td>
<td>100.12x</td>
<td>105.43x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>110</td>
<td>1.58x</td>
<td>1.57x</td>
<td>1.70x</td>
<td>16.88x</td>
</tr>
<tr>
<td>LULESH</td>
<td>26</td>
<td>2.28x</td>
<td>2.27x</td>
<td>2.51x</td>
<td>9.11x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td><strong>3.49x</strong></td>
<td><strong>4x</strong></td>
<td><strong>4.38x</strong></td>
<td><strong>16.86x</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Memory Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Original resident memory in MB</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution via binary tree</th>
<th>Data-centric attribution via shadow memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>230</td>
<td>1.16x</td>
<td>1.17x</td>
<td>1.34x</td>
<td>8.65x</td>
</tr>
<tr>
<td>bzip2</td>
<td>561</td>
<td>1.11x</td>
<td>1.12x</td>
<td>1.12x</td>
<td>8.03x</td>
</tr>
<tr>
<td>gcc</td>
<td>453</td>
<td>15.62x</td>
<td>25.97x</td>
<td>26.03x</td>
<td>36.38x</td>
</tr>
<tr>
<td>h264ref</td>
<td>37</td>
<td>2.49x</td>
<td>2.69x</td>
<td>2.91x</td>
<td>11.47x</td>
</tr>
<tr>
<td>hmer</td>
<td>15</td>
<td>4.38x</td>
<td>4.36x</td>
<td>5.13x</td>
<td>29.39x</td>
</tr>
<tr>
<td>libquantum</td>
<td>96</td>
<td>1.28x</td>
<td>1.30x</td>
<td>1.32x</td>
<td>11.91x</td>
</tr>
<tr>
<td>mcf</td>
<td>1677</td>
<td>1.02x</td>
<td>1.03x</td>
<td>1.03x</td>
<td>6.53x</td>
</tr>
<tr>
<td>omnetpp</td>
<td>170</td>
<td>1.87x</td>
<td>2.35x</td>
<td>3.76x</td>
<td>10.54x</td>
</tr>
<tr>
<td>Xalan</td>
<td>419</td>
<td>25.86x</td>
<td>38.60x</td>
<td>39.12x</td>
<td>46.63x</td>
</tr>
<tr>
<td>ROSE</td>
<td>380</td>
<td>64.38x</td>
<td>98.15x</td>
<td>100.12x</td>
<td>105.43x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>110</td>
<td>1.58x</td>
<td>1.57x</td>
<td>1.70x</td>
<td>16.88x</td>
</tr>
<tr>
<td>LULESH</td>
<td>26</td>
<td>2.28x</td>
<td>2.27x</td>
<td>2.51x</td>
<td>9.11x</td>
</tr>
<tr>
<td><strong>GEOMEAN</strong></td>
<td><strong>3.49x</strong></td>
<td><strong>4x</strong></td>
<td><strong>4.38x</strong></td>
<td><strong>16.86x</strong></td>
<td></td>
</tr>
</tbody>
</table>
# Memory Overhead of CCTLib

<table>
<thead>
<tr>
<th>Program</th>
<th>Original resident memory in MB</th>
<th>Call path on each memory access instruction</th>
<th>Call path on each instruction</th>
<th>Data-centric attribution via binary tree</th>
<th>Data-centric attribution via shadow memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>astr</td>
<td>230</td>
<td>1.16x</td>
<td>1.17x</td>
<td>1.34x</td>
<td>8.65x</td>
</tr>
<tr>
<td>bzip2</td>
<td>561</td>
<td>1.11x</td>
<td>1.12x</td>
<td>1.12x</td>
<td>8.03x</td>
</tr>
<tr>
<td>gcc</td>
<td>453</td>
<td>15.62x</td>
<td>25.97x</td>
<td>26.03x</td>
<td>36.38x</td>
</tr>
<tr>
<td>h264ref</td>
<td>37</td>
<td>2.49x</td>
<td>2.69x</td>
<td>2.91x</td>
<td>11.47x</td>
</tr>
<tr>
<td>hmer</td>
<td>15</td>
<td>4.38x</td>
<td>4.36x</td>
<td>5.13x</td>
<td>29.39x</td>
</tr>
<tr>
<td>libquantum</td>
<td>96</td>
<td>1.28x</td>
<td>1.30x</td>
<td>1.32x</td>
<td>11.91x</td>
</tr>
<tr>
<td>mcf</td>
<td>1677</td>
<td>1.02x</td>
<td>1.03x</td>
<td>1.03x</td>
<td>6.53x</td>
</tr>
<tr>
<td>omnetpp</td>
<td>170</td>
<td>1.87x</td>
<td>2.35x</td>
<td>3.76x</td>
<td>10.54x</td>
</tr>
<tr>
<td>Xalan</td>
<td>419</td>
<td>25.86x</td>
<td>38.60x</td>
<td>39.12x</td>
<td>46.63x</td>
</tr>
<tr>
<td>ROSE</td>
<td>380</td>
<td>64.38x</td>
<td>98.15x</td>
<td>100.12x</td>
<td>105.43x</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>110</td>
<td>1.58x</td>
<td>1.57x</td>
<td>1.70x</td>
<td>16.88x</td>
</tr>
<tr>
<td>LULESH</td>
<td>26</td>
<td>2.28x</td>
<td>2.27x</td>
<td>2.51x</td>
<td>9.11x</td>
</tr>
<tr>
<td>GEOMEAN</td>
<td></td>
<td>1.71x</td>
<td>1.77x</td>
<td>1.99x</td>
<td>11.28x</td>
</tr>
</tbody>
</table>
CCTLib is Scalable

CCTLib overhead of n threads:

\[ OH(n) = \frac{R_c(n)}{R_o(n)} \]

CCTLib scalability for n threads:

\[ S(n) = \frac{OH(1)}{OH(n)} \]

Higher scalability is better, 1.0 is ideal
CCTLib is Scalable

CCTLib overhead of n threads:

\[ OH(n) = \frac{R_c(n)}{R_o(n)} \]

CCTLib scalability for n threads:

\[ S(n) = \frac{OH(1)}{OH(n)} \]

Higher scalability is better, 1.0 is ideal

CCTLib scalability on LAMMPS
CCTLib is Scalable

CCTLib overhead of n threads:

\[ OH(n) = \frac{R_c(n)}{R_o(n)} \]

CCTLib scalability for n threads:

\[ S(n) = \frac{OH(1)}{OH(n)} \]

Higher scalability is better, 1.0 is ideal
Conclusions

• Enhance diagnostic capabilities of fine-grained execution monitoring tools by associating each
  ✦ Instruction ➞ calling context (code-centric attribution)
  ✦ Memory address ➞ data object (data-centric attribution)

• Ubiquitous calling context collection and data-centric attribution is expensive (both memory and time)

• **CCTLib**
  ✦ Provides calling context for Pin tools
  ✦ Achieves ubiquitous code- and data-centric attribution via appropriate choice of algorithms and data structures

• **CCTLib** enables efficient construction of Pin tools that need detailed attribution of costs to contexts or data
CCTLib Enhances Tool Usability
CCTLib Enhances Tool Usability

Inventor

User

https://code.google.com/p/cctlib/