HardBound: Architectural Support for Spatial Safety of the C Programming Language

Joe Devietti*, Colin Blundell†, Milo Martin†, Steve Zdancewic†

*University of Washington, †University of Pennsylvania

devietti@cs.washington.edu, {blundell, milom, stevez}@cis.upenn.edu
This work licensed under the Creative Commons Attribution-Share Alike 3.0 United States License

• You are free:
  • to Share — to copy, distribute, display, and perform the work
  • to Remix — to make derivative works

• Under the following conditions:
  • Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  • Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

• For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to:
  http://creativecommons.org/licenses/by-sa/3.0/us/

• Any of the above conditions can be waived if you get permission from the copyright holder.
• Apart from the remix rights granted under this license, nothing in this license impairs or restricts the author's moral rights.
Who cares about spatial safety, anyway?
HardBound: Spatial Safety for C

- Bounded pointer primitive in hardware
- A new hardware/software contract for pointers
  - Software finds where pointers are created
  - Hardware checks and propagates pointers
- Inspired by software-only fat pointer proposals
  - Pointers become (pointer, base, bound) triples
  + Can provide complete spatial safety
  - Changes memory layout
- Hardware support enables:
  - Unchanged memory layout → high compatibility
  - Efficient fat pointer encodings → low overheads (5%)
 Spatial Violation Example

```c
struct BankAccount {
    char acctID[3];
    int balance;
} b;
...
b.balance = 0;
char* id = b.acctID;
inputID(id);
...
void inputID(char* p) {
    while (*p = readchar())
        p++;
}
```
Preventing Spatial Violations

- treat
- language
- address
- use a safe language
- add bounds checking to C
- all but existing code?
- what about complete
## Software-Only Bounds Checking for C

<table>
<thead>
<tr>
<th>approach</th>
<th>can handle sub-objects?</th>
<th>compatibility?</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>object lookup</td>
<td>yes</td>
<td>low</td>
<td>SafeC [Austin et al, 1994]</td>
</tr>
<tr>
<td>fat pointer</td>
<td>no</td>
<td>high</td>
<td>HardBound – Joe Devietti – ASPLOS 2008</td>
</tr>
</tbody>
</table>
Code Example with Fat Pointers

```c
struct BankAccount {
    char acctID[3];
    int balance;
} b;
...
b.balance = 0;
char* id = b.acctID;
inputID(id);
...
void inputID(char* p) {
    while (*p = readchar())
        p++;
}
```
CCured

+ Uses fat pointers only where needed
  • determined by whole-program type inference
  • significantly reduces overhead

− Curing a program requires programmer help
  − Interfacing with non-cured libraries
  − C nastiness: arbitrary casts, unions, ...

CCured team: 1% of source code lines changed
  • Spec95, Olden, Ptdist

Curing a program is a non-trivial task
HardBound: A New Hope

• Bounded pointer hardware primitive
  • implemented via shadow memory and registers
• Compiler **identifies** where pointers are created
  • examples: malloc, &variable (intraprocedural analysis)
  • using **setbound** instruction
• Hardware **checks** and **propagates** the pointers

+ Fat pointer representation **hidden** by hardware
  + compatible (memory layout unchanged)
  + enables optimized metadata encodings
Rest of Talk

- How do we make HardBound work?
  - pointer propagation and checking
  - memory layout
- How do we make HardBound fast?
  - compressing non-pointers
  - compressing pointers
Metadata Propagation and Checking

- All registers and memory have metadata
- Example operations:
  - `add R1 ← R2 + imm`
    - `R1.value ← R2.value + imm`
    - `R1.base ← R2.base`
    - `R1.bound ← R2.bound`
  - `load R1 ← Memory[R2]`
    - `assert(R2.base ≤ R2.value < R2.bound)`
    - `R1.value ← Memory[R2.value].value`
    - `R1.base ← Memory[R2.value].base`
    - `R1.bound ← Memory[R2.value].bound`
The HardBound Hardware/Software Approach

```c
struct BankAccount {
    char acctID[3];
    int balance;
} b;
...
b.balance = 0;
char* id;
id = inputID(id);
...
void inputID(char* p) {
    while (*p = readchar())
        p++;
}
```
Virtual Memory

struct {
    int* ptr;
    int i;
} s;

enables compatibility…
…how do we make it fast?
Eliminating Metadata for Non-pointers

• Most data are not pointers

- add 1-bit tag to act as a filter
  - prevents an expensive base/bound lookup
- lives in virtual memory
  - 1 bit per word (3% overhead)
Compressing In-Memory Metadata

1. Many pointers point to the **beginning** of an object
2. Many pointers point to **small** objects

_used opportunistically on stores_

**External** encoding: fold size into tag
Compressing Metadata for Pointers

- Most programs don’t use much of virtual memory
- Pointers contain more bits than are really needed

Internal encoding: hide offset in pointer

pointer: 1000  base: 1000  bound: 1003

HardBound Metadata Lookup

Processor Core

tag$

TLB

d$

TLB

i$

L2$

Memory
Experiments

• Experimental methodology
  • Source-to-source compiler written in CIL
  • FeS$_2$ x86 inorder single-cycle processor with Simics

• Tested correctness with a suite of 291 spatial violations [Kratkiewicz & Lippmann, 2005]
  • 286 are compatible with our simulation environment
  • no false positives, no false negatives
Performance Experiments

• Olden pointer-intensive benchmark suite
  • no source code modifications required for correctness
    • we even found a bug in em3d
  • we made two small performance-only modifications
    • bh: manually inlined two functions
      • reduce number of redundant setbound insns
    • mst: manually inserted three setbound insns
      • to tighten bounds for better compression
## Experimental Configurations

<table>
<thead>
<tr>
<th>encoding</th>
<th>size of tag space</th>
<th>tag cache</th>
<th>encodes pointers up to</th>
</tr>
</thead>
<tbody>
<tr>
<td>external 4-bit</td>
<td>4 bits</td>
<td>8KB</td>
<td>56 B</td>
</tr>
<tr>
<td>internal 4-bit</td>
<td>1 bit</td>
<td>2KB</td>
<td>56 B</td>
</tr>
<tr>
<td>internal 11-bit</td>
<td>1 bit</td>
<td>2KB</td>
<td>4 KB</td>
</tr>
</tbody>
</table>
Normalized Runtime Results

- additional memory latency
- stalling on pointer metadata
- μops for loading/storing bounds
- setbound instructions

5% average overhead
Conclusions

• HardBound is a hardware/software approach that provides spatial safety for C

• Hardware-managed pointer metadata enables:
  • high compatibility (doesn’t change memory layout)
  • low overhead (enables encoding tricks)

• Our experiments show that HardBound is:
  • effective (no false positives, no false negatives)
  • efficient (compressing metadata works)

• Future work
  • further reduce overhead using CCured’s static analysis
  • temporal safety (dangling pointers)