Prototyping Architectural Support for Program Rollback Using FPGAs

Radu Teodorescu and Josep Torrellas
http://iacoma.cs.uiuc.edu

University of Illinois at Urbana-Champaign

Motivation

- Problem:
  - Software bugs – major cause of system failure
  - Production software is hard to debug
- Continuous debugging is needed
- Software-based dynamic monitoring tools
  - Can catch a wide range of bugs
  - Orders of magnitude slowdowns

Motivation

- Alternative solutions
  - Hardware support for debugging
    - Low overhead
    - Exiting support is still modest
- Our system:
  - Hardware-assisted, lightweight debugger
  - Monitoring, detection and recovery from bugs in production systems

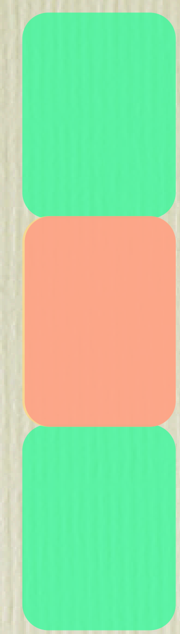
Contributions

- We implemented a hardware prototype of a debugging-aware processor
- We show that simple changes to a general purpose processor can provide powerful debugging primitives
- We run experiments on buggy programs
- Implementation technology: FPGA
  - Ideal platform for rapid prototyping
  - Validate design, measure hardware overheads, run realistic experiments

Debugging Production Code

- Applications run in multiple states:
  - Normal
  - Speculative (can be undone)
  - Re-execute
- Transition between states is controlled by software

Dynamic execution



Debugging Production Code

Original code

```
num=1;
...
p=m[a[*x]]+&y;
...
num++;
```

Instrumented code

```
num=1;
enter_spec();
p=m[a[*x]]+&y;
...
if(pstate()==REEXEC)
{
 info_collect();
}
exit_spec(flag);
num++;
```

Replay

Rollback

Dynamic execution

```
num=1;
enter_spec();
p=m[a[*x]]+&y;
...
if(pstate()==REEXEC)
{
 info_collect();
}
exit_spec(flag);
num++;
```

Normal

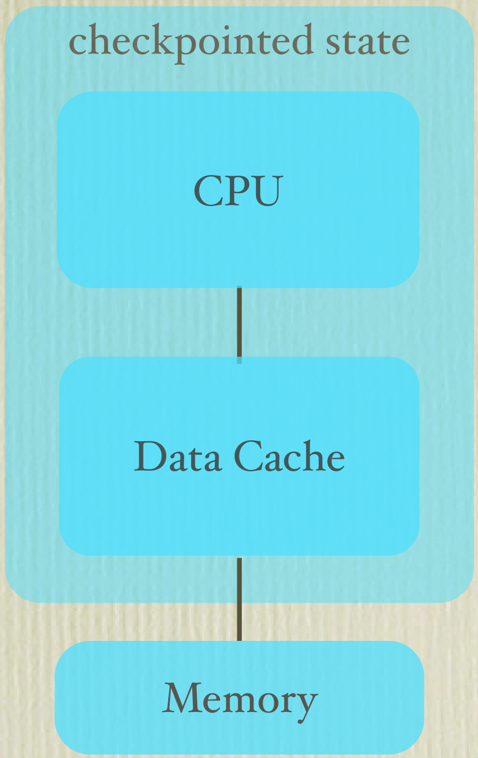
Speculative

Re-execute

System Implementation

Hardware Extensions

- Undo program execution
    - Large code sections
    - Small overhead
    - Software control
- Lightweight checkpointing
- Hardware support needed:
    - Register checkpointing
    - Speculative data cache

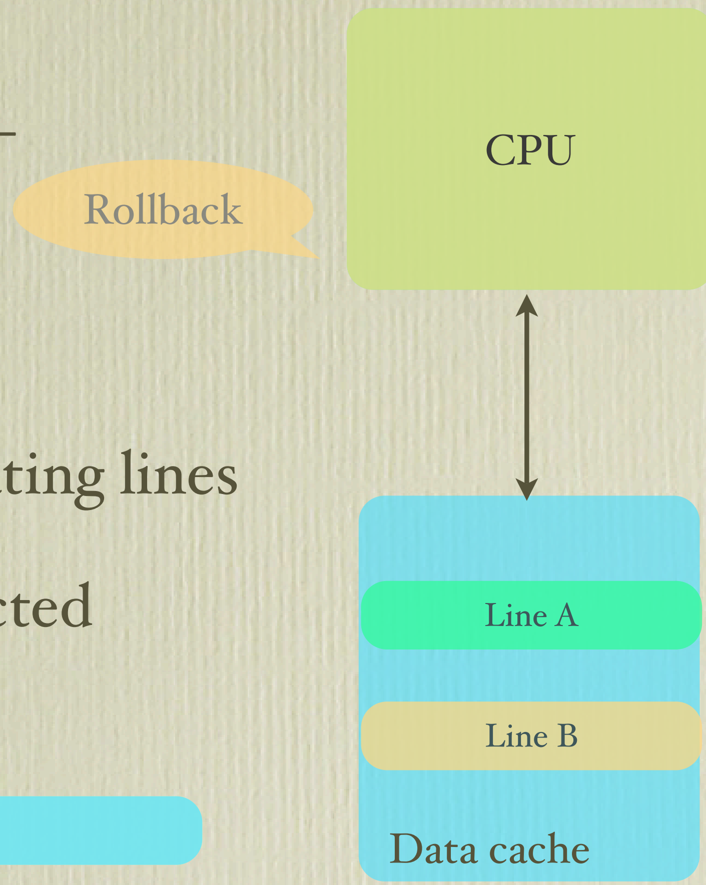


Register Checkpointing

- Needed to allow restoration of processor state
- Beginning of speculative execution
  - Register file is copied into a shadow register file
- End of speculative execution
  - Commit: discard checkpoint
  - Rollback: restore registers & PC from checkpoint

Speculative Data Cache

- Holds both speculative and non-speculative data
- Each line has a “speculative” bit
- Cache walk: merging or invalidating lines
- Speculative lines cannot be evicted

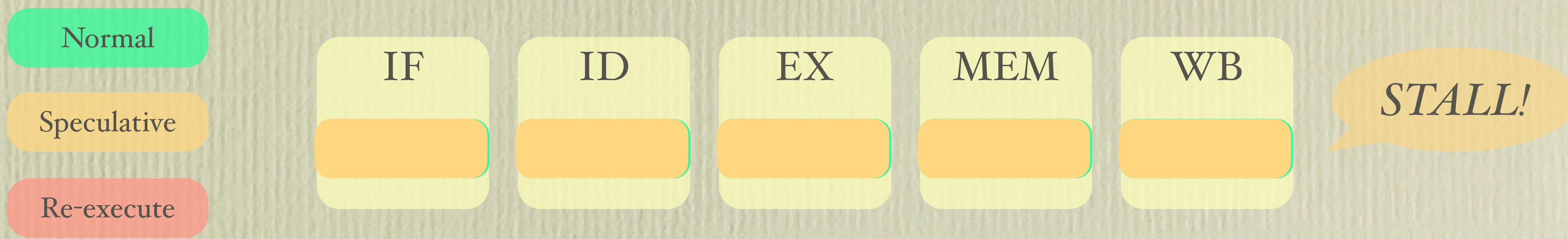


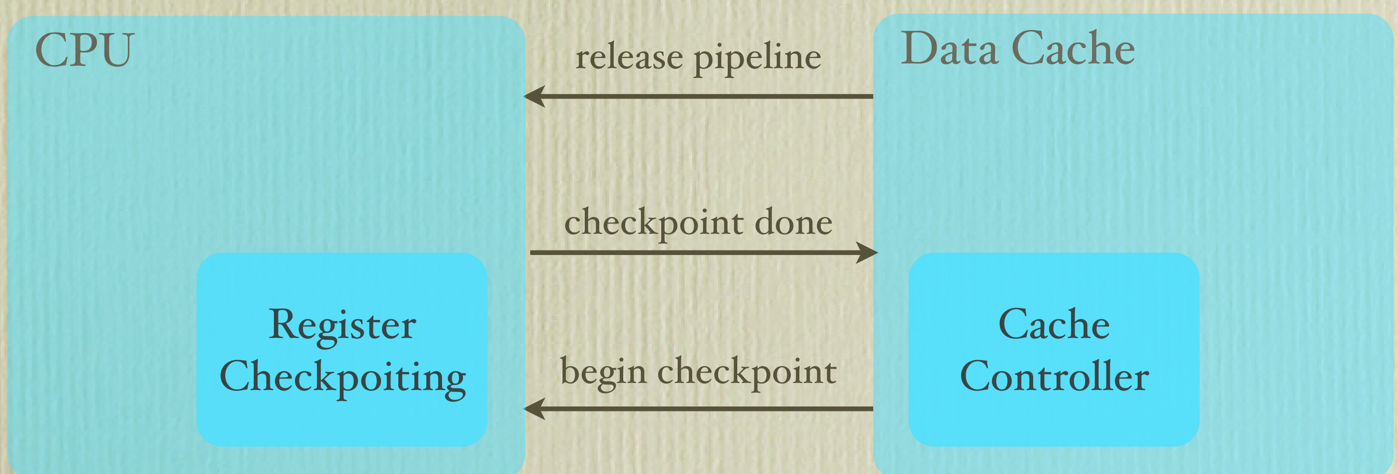
data cache line

Software Control

- Give the compiler control over speculative execution
- Control instructions:
  - Begin speculation
  - End speculation (commit or rollback)
- We use SPARC's special access load
  - `LDA [r0] code, r1`

Begin Speculative Execution





Limits

- Size of the speculative window is affected by:
    - Cache size and associativity - cache overflow
    - I/O operations cannot be rolled back
- In both cases exceptions are raised
    - Early commit
    - OS intervention: buffer speculative state or I/O instructions

Experiments and Results

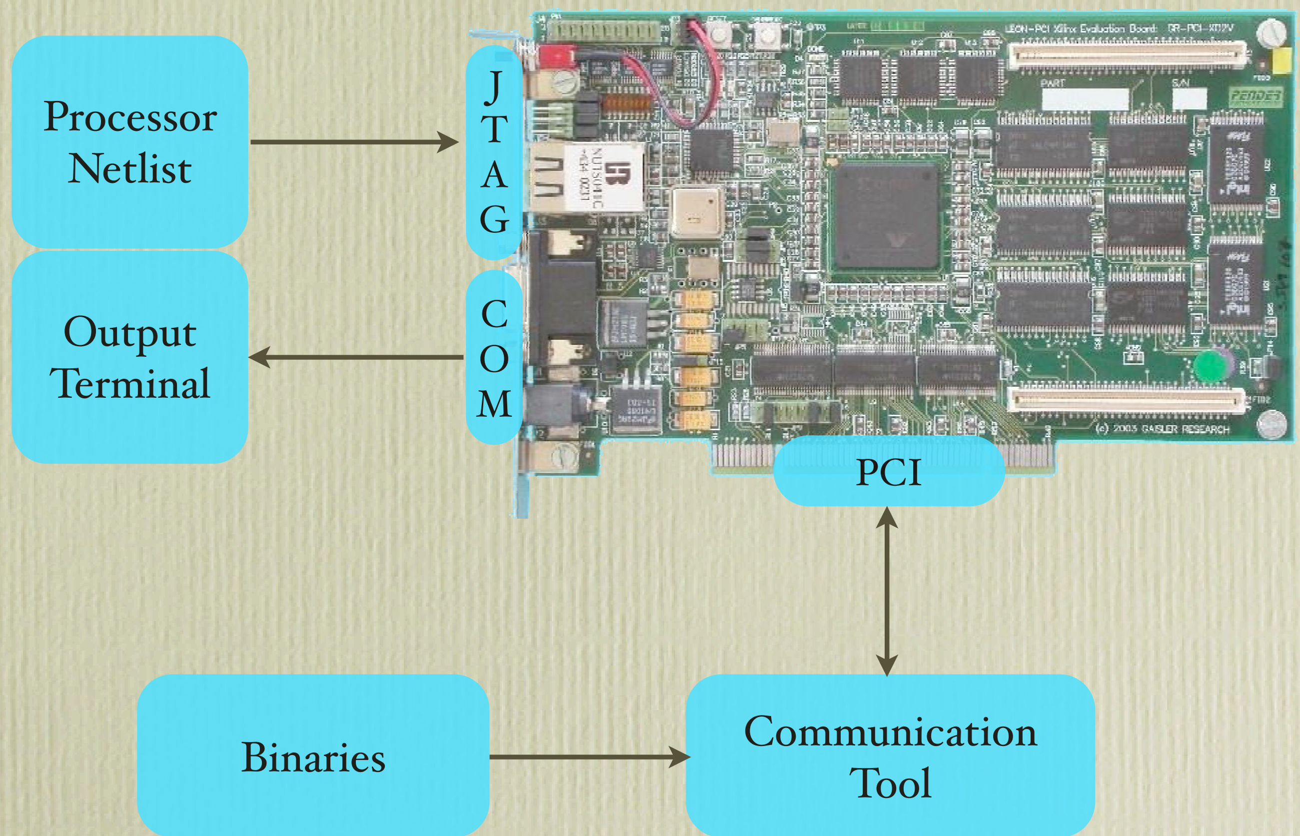
Processor Prototype

- LEON2 - SPARC V8 compliant processor
- Single issue, 5-stage pipeline
- Windowed register file
  - 2-32 sets, 16 registers
- L1 instruction and data caches
  - 1-4 sets, up to 64KB/set
- Synthesizable, open source VHDL code

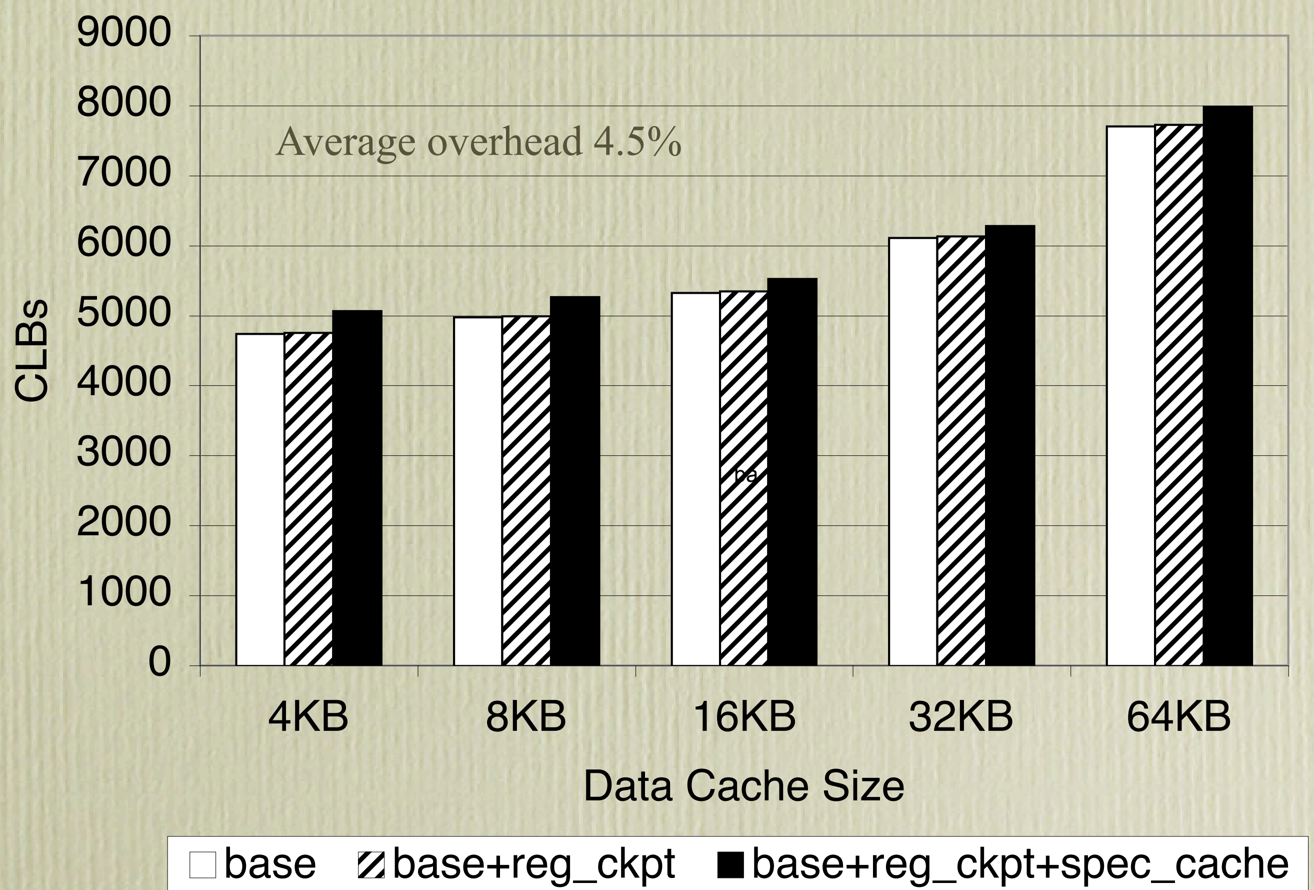
Experimental Infrastructure

- System on a chip: PCI, Ethernet and serial interfaces
- Development tools
  - RTL Simulation - ModelSIM
  - Synthesis - Xilinx ISE 6.1
- Development board:
  - Xilinx Virtex II XC2V3000, 64 Mbytes SDRAM
- Linux embedded

Deployment

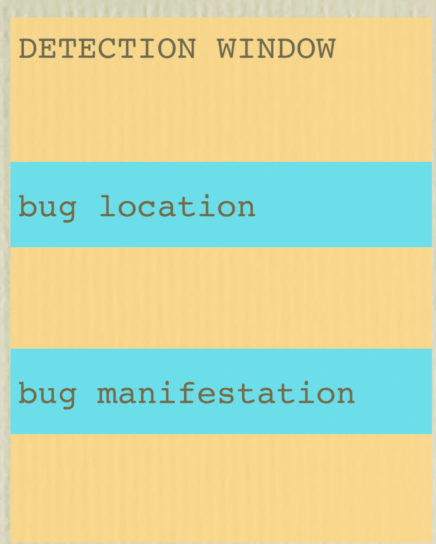

Processor Netlist
JTAG
Output Terminal
COM
PCI
Binaries
Communication Tool

Hardware Overhead
Configurable Logic Blocks



Buggy Applications

- Applications with known bugs
- Manually instrument the code
- Detection window contains:
    - bug location
    - bug manifestation
- Determine if we can roll back the buggy code section
- Test configuration: 32KB data cache, 4KB instruction



Buggy Applications

Application	Bug Description	Successful rollback	Dynamic Instructions
ncompress-4.2.4	Input file name longer than 1024 bytes corrupts stack return address	Yes	10653
polymorph-0.4.0	Input file name longer than 2048 bytes corrupts stack return address	No	103838
tar-1.13.25	Unexpected loop bounds causes heap object overflow	Yes	193
man-1.5h1	Wrong bounds checking causes static object corruption	Yes	54217
gzip-1.2.4	Input file name longer than 1024 bytes overflows a global variable	Yes	17535

Conclusions

- We implemented a hardware prototype of a processor with software controlled speculative execution
- We show that simple changes to a general purpose processor can provide powerful debugging primitives
- Obtained an estimate of the hardware overhead and run experiments on buggy programs
- We are looking at the integration of our hardware with compiler and operating system support

Prototyping Architectural Support for Program Rollback Using FPGAs

Radu Teodorescu and Josep Torrellas
http://iacoma.cs.uiuc.edu

University of Illinois at Urbana-Champaign