FlipBack: Automatic Target Protection Against Soft Errors

Xiang Ni
Parallel Programming Lab
Soft Errors

• Common source of soft errors
  • Electrical noise
  • External radiation
  • Manufacturing fault

• Data corruption: we may or may not know

Shrinking chip size
• More energy efficient
• Higher soft error rate
Soft Errors

• Common source of soft errors
  • Electrical noise
  • External radiation
  • Manufacturing fault

• Data corruption: we may or may not know

Shrinking chip size
• More energy efficient
• Higher soft error rate

Less energy required to corrupt data
Motivation Example

```
ghostMsg  msgsRecvd++  msgsRecvd== expectedMsg

No

Yes
```
Motivation Example

ghostMsg \rightarrow \text{msgsRecvd}++ \rightarrow \text{msgsRecvd}==\text{expectedMsg}

- No
- Yes

\text{expectedMsg}
00000111
Motivation Example

ghostMsg \rightarrow msgsRecvd++ \rightarrow \begin{cases} 
\text{msgsRecvd}==\text{expectedMsg} & \text{No} \\
\text{expectedMsg} & \text{Yes} 
\end{cases}

expectedMsg

00001111
Motivation Example

Expected message: 

- `msgsRecvd == expectedMsg`  
- `ghostMsg: 00001111`  
- `7 -> 15`
Motivation Example

ghostMsg → msgsRecvd++

msgsRecvd==expectedMsg

HANG

No

Yes

expectedMsg

7 → 15

00001111
Motivation Example

ghostMsg → msgsRecvd++

msgsRecvd== expectedMsg

HANG

forced

No

Yes

expectedMsg

7 → 15

00000011
Motivation Example

expectedMsg
msgsRecvd==
ghostMsg
msgsRecvd++
HANG
No
Yes
expectedMsg
7 —> 3
00000011
Motivation Example

Stop accepting messages much earlier: incorrect result

expectedMsg  7 → 3
00000011
Runtime Guided Replication
Runtime Guided Replication

- Control Variables
  - msgsRecvd, expectedMsg
  - Affecting program flow
Runtime Guided Replication

• Control Variables
  • msgsRecvd, expectedMsg
• Affecting program flow
• How do we ensure the program control flow is correct?
  • Fully duplication is expensive: less than 50% resource utilization or at least twice the running time
Runtime Guided Replication

- Control Variables
  - msgsRecvd, expectedMsg
  - Affecting program flow
- How do we ensure the program control flow is correct?
  - Fully duplication is expensive: less than 50% resource utilization or at least twice the running time
- What about only duplicating the computation that affects program flow?
  - Leverage a compiler slicing pass
  - Reduce computation time
  - Avoid doubling the memory
Input:
f: the targeted function to perform slicing on
c: set of control variables
Output: slices: the program slice for recomputation

// search for slicing criterions

foreach Instruction I in f do
  if Defs(I) ∩ c or I sends messages then
    criterions.push(I);
  end
end

while !criterions.empty() do
  I ← criterions.top(); criterions.pop();
  if !I.processed() then
    slices.push(I);
    // data flow analysis
    foreach Values I' in Uses(I) do
      foreach Instruction I'' in Defs(I') do
        if I'' may lead to I then
          criterions.push(I'');
        end
      end
    end
    // control flow analysis
    foreach BasicBlock B that may lead to I do
      criterions.push(B.getTerminator());
    end
  end
end
Input:
f: the targeted function to perform slicing on
c: set of control variables

Output: slices: the program slice for recomputation
// search for slicing criterions

```c
void Stencil::beginNextIter() {
    iterCount++;
    if(iterCount >= totalIter){
        mainProxy.done(); //program exits
    }else{
        for(int i = 0; i < totalDirections; i++) {
            ghostMsg * m = createGhostMsg(dirs[i]);
            copy(m->data, boundary[i]);
            int sendTo = myIdx+dirs[i];
            stencilProxy(sendTo).receiveMessage(m);
        }
    }
}
```

// data flow analysis
```c
foreach Values I' in Uses(I) do
    foreach Instruction I'' in Defs(I') do
        if I'' may lead to I then
            criterions.push(I'');  
    criteria.push(I);
```

// control flow analysis
```c
foreach BasicBlock B that may lead to I do
    criterions.push(B.getTerminator());
```
```c
end
```
void Stencil::beginNextIter() {
    iterCount++;
    if(iterCount >= totalIter){
        mainProxy.done(); //program exits
    }else{
        for(int i = 0; i < totalDirections; i++) {
            ghostMsg * m = createGhostMsg(dirs[i]);
            copy(m->data, boundary[i]);
            int sendTo = myIdx+dirs[i];
            stencilProxy(sendTo).receiveMessage(m);
        }
    }
}
void Stencil::beginNextIter() {
    iterCount++;
    if(iterCount >= totalIter){
        mainProxy.done(); //program exits
    }else{
        for(int i = 0; i < totalDirections; i++) {
            ghostMsg * m = createGhostMsg(dirs[i]);
            copy(m->data, boundary[i]);
            int sendTo = myIdx+dirs[i];
            stencilProxy(sendTo).receiveMessage(m);
        }
    }
}

Compiler Slicing Pass

Input:
f: the targeted function to perform slicing on
\( c \): set of control variables
\textbf{Output}: slices: the program slice for recomputation
// search for slicing criterions
\begin{verbatim}
foreach Instruction I in f do
    if Defs(I) \subset c or I sends messages then
        criterions.push(I);
end
while !criterions.empty() do
    I\leftarrow\text{criterions.top()}; criterions.pop();
    if !I.processed() then
        slices.push(I);
        // data flow analysis
        foreach Values I' in Uses(I) do
            foreach Instruction I'' in Defs(I') do
                if I'' may lead to I then
                    criterions.push(I'');
            end
        end
    end
end
// control flow analysis
foreach BasicBlock B that may lead to I do
    criterions.push(B.getTerminator());
end
end
\end{verbatim}
void Stencil::beginNextIter() {
    iterCount++;
    if(iterCount >= totalIter) {
        mainProxy.done(); // program exits
    } else {
        for(int i = 0; i < totalDirections; i++) {
            ghostMsg * m = createGhostMsg(dirs[i]);
            copy(m->data, boundary[i]);
            int sendTo = myIdx+dirs[i];
            stencilProxy(sendTo).receiveMessage(m);
        }
    }
}

// search for slicing criterions
foreach Instruction I in f do
    ifDefs(I) ⊂ c or I sends messages then
        criterions.push(I);
end

while !criterions.empty() do
    I ← criterions.top(); criterions.pop();
    if !I.processed() then
        slices.push(I);
        // data flow analysis
        foreach Values I' in Uses(I) do
            foreach Instruction I'' in Defs(I') do
                if I'' may lead to I then
                    criterions.push(I'');
                end
            end
        end
        // control flow analysis
        foreach BasicBlock B that may lead to I do
            criterions.push(B.getTerminator());
        end
    end
end

Input:
f: the targeted function to perform slicing on
c: set of control variables
Output: slices: the program slice for recomputation
The Role of Runtime System

• Creation of shadow shares
The Role of Runtime System

- Creation of shadow chares
  - Initialize with the same control variables from the original chare
The Role of Runtime System

• Creation of shadow chares
  • Initialize with the same control variables from the original chare
  • Share the same pointers of the non-control variables
The Role of Runtime System

- Creation of shadow chares
  - Initialize with the same control variables from the original chare
  - Share the same pointers of the non-control variables
  - Compare the values of control variables and outgoing messages at the end of entry method
Runtime Guided Replication

Input: o: the original full fledged chare
s: the shadow chare

// RTS receives a message M for o
1 checkpointControl(o);
2 checkpointControl(s);
3 restart ← true;
4 while restart do
   // buffering outgoing messages
5   o.invoke(M); s.invoke(M);
6   if compareControl(o, s) and compareMsgs(o, s) then
7       restart ← false;
8       sendMsgs(o); deleteMsgs(s);
9    end
10 else
11    restartControl(o); restartControl(s);
12 end
13 end
Another Example

```java
void Stencil:invokeComputation() {
    //computation routine
    for (int i = 0; i < size; ++i) {
        temperature[i] = ...;
    }
}
```

- The previous method fails to protect loop index `i`
- Lifetime ends before the end of the entry method
- However, if bit flip occurs to `i`: incorrect data to be used or program crashes
Selective Instruction Duplication

1 ;label 0
2 %1 = add i, 1
3 %2 = add i, 1
4 %3 = icmp eq %1, %2
5 br %3, label %4, label %6
6 ;label 4
7 5 = add i, 1
8 br label %6
9 ;label 6
10 %7 = phi [%1, label %0], [%5, label %4]
Protection for Field Data

- The rule holds in nature also be held in scientific programs

Stencil2d

OpenAtom
Protection for Field Data
Protection for Field Data

- Spatial similarity
• Spatial similarity
Protection for Field Data

• Spatial similarity

• Temporal similarity
Protection for Field Data

- Spatial similarity

\[ d(i-1,j-1), d(i,j), d(i-1,j+1), d(i+1,j+1), d(i+1,j), d(i-1,j), d(i-1,j-1), d(i+1,j-1) \]

- Temporal similarity

  - data at time step t-2k, t-k, t
Protection for Field Data

• Spatial similarity

• Temporal similarity
  • data at time step t-2k, t-k, t

• Spatial temporal similarity
Protection for Field Data

- Spatial similarity

- Temporal similarity
  - data at time step t-2k, t-k, t

- Spatial temporal similarity
  - spatial similarity of temporal updates
Protection for Field Data

- Spatial similarity

- Temporal similarity
  - data at time step t-2k, t-k, t

- Spatial temporal similarity
  - spatial similarity of temporal updates
  - temporal similarity of spatial differences
Evaluation

• **Miniaero**
  • Mantevo mini-applications suite
  • compressible Navier-Stokes equations using explicit RK4 method

• **Particle-in-cell**
  • Intel PRK benchmark suite
  • Charm++ implementation
  • Particles are distributed within a fixed grid of charges. At each time step, PIC calculates the impact of the Coulomb potential of particles with related grid points.

• **Stencil3d**
  • 7-point stencil-based computation on a 3D-structured mesh

• Fault Injection with **LLFI**
  • random time
  • random processor
Evaluation

Miniaero
Evaluation

Particle-in-cell

(a) Original: control
(b) Protected: control
(c) Original: communication
(d) Protected: communication
(e) Original: computation (integer)
(f) Protected: computation (integer)
(g) Original: computation (floating point)
(h) Protected: computation (floating point)
Evaluation

Stencil3d

(a) Original: control
(b) Protected: control
(c) Original: communication
(d) Protected: communication
(e) Original: computation (integer)
(f) Protected: computation (integer)
(g) Original: computation (floating point)
(h) Protected: computation (floating point)
Discussion

• Compare with traditional checkpoint/restart strategy
Discussion

• Compare with traditional checkpoint/restart strategy
  
  • For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution
• Compare with traditional checkpoint/restart strategy

  • For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution

  • At the cost of global restart
• Compare with traditional checkpoint/restart strategy
  • For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution
  • At the cost of global restart
  • With FlipBack, overhead of local restart is minimal
Discussion

• Compare with traditional checkpoint/restart strategy

• For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution

• At the cost of global restart

• With FlipBack, overhead of local restart is minimal
Discussion

• Compare with traditional checkpoint/restart strategy

• For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution

• At the cost of global restart

• With FlipBack, overhead of local restart is minimal
Conclusion

• Leverage compiler and runtime techniques for a cheaper way to protect applications against silent data corruptions

• Almost 100% coverage

• 6-20% overhead