LetGo: A Lightweight Continuous Framework for HPC Applications under Failures

Bo Fang, Qiang Guan, Nathan Debardeleben, Karthik Pattabiraman and Matei Ripeanu
Resilience is a critical factor for system to scale.

\[ MTBF = \frac{\text{Total Time}}{\text{Number of failures}} \]

\[ MTBF_{\text{system}} = \frac{MTBF_{\text{node}}}{\text{Number of nodes}} \]

Fault tolerance techniques are needed!
Classical FT Technique: Checkpoint/Restart

\[ \text{Efficiency} = \frac{\Sigma(Ckpt\ interval)}{\Sigma(Ckpt\ interval + Overhead)} \]

Optimal checkpoint interval = \( f(Ckpt\ overhead, MTBF) \)  

[Young, Daly]
Design Space for Recovery strategies

- **Application-aware**
  - Rollback: [Gamel2013]
  - Forward: [Wu2016]

- **Application-agnostic**
  - Rollback: [Aupy2013]
    - Classical system-level C/R
  - Forward: [Chien2015]
    - LetGo
    - [Bo2017]
Why this could work?

Can we detect the error?

After the state is repaired …

Will the crash happen again?

Can we expect correct results?

Can we detect incorrect results?

OS Exceptions

Corrupted state is contained [Li2015]

Applications with convergent behavior

App-specific correctness check
Our Approach: LetGo

Design → Evaluation Methodology → Evaluation Results
Forward Recovery with LetGo

Expected Benefits
- **Lower checkpoint frequency**
- Avoid the overhead of restart

Potential risk:
- Higher rate of incorrect results (SDCs)
LetGo – Design Requirements

Transparency
- No need to change the application code

Convenience
- No need to recompile the application or make changes to the runtime environment

Low overhead

Low rate of newly introduced failures
LetGo - Design Overview

LetGo

Modifier

Monitor

Process State

Signal Handling

Process execution

OS
Monitor

Intercepts OS signals for applications
- Uses GDB to attach to application at launch
- Configures the signal handling (SIGSEGV, SIGBUS, SIGABORT)

Benefits:
- No assumption about the compilation level
- No change to the application
- Low overhead (0.1%)
Modifier

Goal: Increase the probability of successful continued execution (low rate of newly introduced failures)

Basic action: move the program counter (PC) after the crash causing instruction

Optional: Heuristics to repair state
Modifier: example

- A bit flip affects the computation logic

```
push %rbp
mov %rsp, %rbp
sub $0x30, %rsp
movsd, -0x28(%rbp), %xmm2
subsd %xmm0, %xmm2
movq %xmm2, %rax
mov %rax, -0x20(%rbp)
```
Evaluation questions

Can we detect the error?

After the state is repaired

Will the crash happen again?

Can we expect correct results?

Can we detect incorrect results?
Evaluation questions

- Q1: What is the chance of the continued execution?
- Q2: What is the chance of successful state repair leading to results?
- Q3: What is the [increase in the] rate of (un)detected incorrect results.
- Q4: What is the performance improvement LetGo brings?

Evaluation methodology

- Fault injection: Q1, Q2, Q3
- Simulations: Q4
## Benchmarks

<table>
<thead>
<tr>
<th>Application</th>
<th>Domain</th>
<th># of dyn inst. (billions)</th>
<th>Data to check for SDC</th>
<th>Criteria used in application check</th>
</tr>
</thead>
<tbody>
<tr>
<td>LULESH</td>
<td>Hydrodynamics</td>
<td>1.0</td>
<td>Mesh</td>
<td>Number of iterations: exactly the same</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Final origin energy: correct to at least 6 digits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Measures of symmetry: smaller than 10⁻⁸</td>
</tr>
<tr>
<td>CLAMR</td>
<td>Adaptive mesh refinement</td>
<td>2.8</td>
<td>Mesh</td>
<td>Threshold for the mass change per iteration</td>
</tr>
<tr>
<td>COMD</td>
<td>Classical molecular dynamics</td>
<td>5.1</td>
<td>Each atom’s property</td>
<td>Energy conservation</td>
</tr>
<tr>
<td>SNAP</td>
<td>Discrete ordinates transport</td>
<td>1.6</td>
<td>Flux solution</td>
<td>The flux solution output should be symmetric</td>
</tr>
<tr>
<td>PENNANT</td>
<td>Unstructured mesh physics</td>
<td>1.7</td>
<td>Mesh</td>
<td>Energy conservation</td>
</tr>
<tr>
<td>HPL</td>
<td>Dense linear solver</td>
<td>1.2</td>
<td>Solution vector</td>
<td>Residual check on the solution vector</td>
</tr>
</tbody>
</table>
Fault Injection Methodology

**Profile**

1. Randomly pick an instruction
2. Flip a bit in the target register
3. Observe outcome

**Outcome**
- Crash: OS exception
- SDC: Undetected incorrect results
- Benign: Correct results
- Detected: Detected incorrect results

LetGo works at crashes
Evaluation Results
(average over all benchmarks with convergent behavior)

LetGo allows 62% of crashes continue to execute with marginal increase in producing SDCs. This indicates that with LetGo, the system level MTBF can be increased.
Evaluation questions

- Q1: What is the chance of the continued execution?
- Q2: What is the chance of successful state repair leading to results?
- Q3: What is the [increase in the] rate of (un)detected incorrect results.
- Q4: What is the performance improvement LetGo brings? What is the impact of system scale?
Simulation Results

- **Efficiency** = *Useful time*/*Total time*

1. As checkpoint overheads increase, LetGo performance advantage increases
2. As the system scales, the LetGo performance advantage increases
Conclusion

- Can we detect the error?
  - Yes
- After the state is repaired
- Will the crash happen again?
  - 62% Not crash
- Can we expect correct results?
  - 54% Correct
- Can we detect incorrect results?
  - 3% detected
Conclusion

- LetGo improves the overall C/R efficiency and leads to a longer checkpoint interval.
- LetGo offers “clean” solution for HPC environment

- Email: bof@ece.ubc.ca
- Github: https://github.com/flyree/LetGo.git
- Netsyslab UBC: http://netsyslab.ece.ubc.ca/
- Dependability lab UBC: http://blogs.ubc.ca/karthik/
Modifier: example

push %rbp
mov %rsp, %rbp
sub $0x30, %rsp
movsd, -0x28(%rbp), %xmm2
subsd %xmm0, %xmm2
movq %xmm2, %rax
mov %rax, -0x20(%rbp)

bit-flip, producing an invalid address

Challenge I: data corruption

... movsd, -0x28(%rbp), %xmm2 movsd, $0x0, %xmm2 subsd %xmm0, %xmm2 ...

%xmm2 may contain random value

Challenge II: pointer corruption

push %rbp
mov %rsp, %rbp
sub $0x30, %rsp ...

%rbp should be within [%rsp, %rsp+30]
State Machine of a C/R with LetGo

- **COMP**: Application runs in a checkpoint interval
- **VERIF**: Application checks the correctness
- **CHK**: Application takes checkpoint
- **LETGO**: LetGo fixes the crash
- **CONT**: Application continues after the fix

Design Evaluation
Methodology
Result
Computation Convergence Example