A PROGRAMMING MODEL FOR RESILIENCE IN EXTREME SCALE COMPUTING

Saurabh Hukerikar, Pedro C. Diniz, Robert F. Lucas
Introduction

- In supercomputing systems today hardware errors account for up to 60% of the total failures;
  - 40% attributed to memory related failures
- Computing platforms abstract lower layers of computing stack to provide application with correct execution
- Many algorithms are inherently resilient to errors
- Programmers may have fault tolerance knowledge but no convenient mechanisms to convey this knowledge to the system
Overview

Resiliency Oriented Programming Model Extensions
Evolutionary Approach: Based on current, familiar language constructs

Multi-Layer Resiliency Approach
Engage fault tolerance knowledge from all software layers including
- Application
- Compiler infrastructure
- Programming language features
- Operating system

Fault Model
Multi-bit memory errors uncorrectable by ECC schemes
Programming Extensions for Resilience

- Type Declarations
- Dynamic Memory Allocation
- Code Sections
- Looping constructs
- Procedure calls
Programming Model Extension

Tolerant Type Qualifiers

tolerant int rgb[XDIM][YDIM];

tolerant<MAX.VALUE=...> unsigned int counter;

tolerant<precision.6f> double low_precision;
Programming Model Extension

Tolerant Malloc

```
<type>* <var> = (cast) tolerant_malloc(sizeof(<type>));

<type>* <var> = (cast) tolerant_malloc(NUM * sizeof(<type><MAX.VALUE=..>));
```

- Managing tolerant areas on the heap space
Programming Model Extension

Tolerant Preprocessor Directives

```plaintext
#tolerant begin
<code>
...
...
...
#tolerant end
```

- Tolerant Regions of Code
void pi() {
    int i;
    double step, x, sum = 0.0;
    double pi;
    step = 1.0/N;

tolerant_for( i =0; i < 10000000; i++; N/3 )
{
    x = ( i + 0.5 ) * step ;
    sum += 4.0/(1.0 + x*x) ;
}
pi = sum* step ;
}
Programming Model Extension

Tolerant Procedures

tolerant void residual ( tolerant<MAX.VAL=32> int arg1,
    tolerant int arg2)
{
    tolerant int local_var1;
    tolerant int local_var2;
    ...
}

Allows programmer to specify tolerance on the stack addresses
• Iterative refinement
• Deep recursion algorithm
System Workflow
## Results
### HPCC Random Access

<table>
<thead>
<tr>
<th>Table Size</th>
<th>Entries</th>
<th>Bytes</th>
<th>Random Updates</th>
<th>Faults Injected</th>
<th>Faults Survived</th>
<th>% Faults Survived</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^18 entries</td>
<td>262,144</td>
<td>262,144</td>
<td>262,144</td>
<td>6</td>
<td>5</td>
<td>83.33</td>
</tr>
<tr>
<td>2^20 entries</td>
<td>4,194,304</td>
<td>4,194,304</td>
<td>4,194,304</td>
<td>26</td>
<td>24</td>
<td>92.31</td>
</tr>
<tr>
<td>2^22 entries</td>
<td>16,777,216</td>
<td>16,777,216</td>
<td>16,777,216</td>
<td>99</td>
<td>97</td>
<td>97.97</td>
</tr>
</tbody>
</table>
Summary

- Our approach demonstrates the benefits of programmer interfaces to express fault tolerance knowledge to the lower levels of system abstraction

Future Directions and Ongoing Work

- Richer set of interfaces and opportunities for cross-layer resilience
- Explore more error models for extreme scale systems
- Self Healing
Information Sciences Institute

Saurabh Hukerikar, Pedro C. Diniz, Robert F. Lucas
{saurabh, pedro, rflucas}@isi.edu