ARGOS Experiment Software

ARGOS Experiment Software: Putting It All Together
by Philip P. Shirvani
RATS
October 4, 1999

Outline
- Introduction
- Experiment Setup
- Errors and Their Effects
- Software Modules
- Error Detection and Recovery
- Status
- Future Work

ARGOS Project [1]
- Rad-Hard vs. COTS Hardware
- Real Space Environment
- Only Software Tech. in COTS Board
- Collect Data
  - Processor and memory errors
  - Software error detection tech.
- Determine Tradeoffs
  - Fault avoidance
  - Fault tolerance

Computing Testbed [2]
- Rad-Hard Board
  - Harris RH3000 radiation-hardened chip set
  - Self-checking pair
  - SOI SRAMs
  - EDAC for main memory
- COTS Board
  - IDT R3081
  - No error detection hardware
  - No EDAC
  - SRAM-based FPGA

Requirements
- Error Detection
  - Programs to exercise functional units
  - Software-implemented error detection added
  - Maximize error detection coverage
- Error Collection
  - Log type, time, position, etc.
  - Correct transmission of log
- Error Recovery
  - Automatically

Errors
- Main Type of Errors
  - Radiation-induced transient errors
    - e.g., Alpha particles and cosmic rays
  - Single-Event Upsets (SEUs)
  - Multiple-bit error caused by a single SEU
- Main Locations
  - Memory and processor
SEU Effects on Programs
- Bit-Flips in Memory
  - Code segments
  - Data segments
- SEU in Processor
  - Computation error
  - Control-flow error
  - Load/store error

Error Detection
- Rad-Hard Board
  - Memory errors
    - EDAC
  - Processor errors
    - Mostly detected by self-checking pair
    - Coverage enhanced by software detection
- COTS Board
  - All software (OS and applications)

Memory Scrubbing
- Rad-Hard Board
  - Hardware EDAC
    - Checked upon memory read
    - Periodic sweeping of memory
    - Dummy task
    - Preventing accumulation of errors
- COTS Board
  - Software-implemented EDAC [3][4]
  - Correct bit-flips in code segments
  - Periodic or on-demand option

Software Error Detection
- Time Redundancy
  - Stutter-Step Mode execution (SSM) [5-8]
  - Software duplication/TMR [9]
- Control Flow Checking
  - Signature Analysis by Instr.s (SAI) [10-12]
  - Watchdog task and timers [13]
- Other
  - Algorithm-Based Fault Tolerance (ABFT) [10]
  - Assertions
  - Programming practices [13][14]

Error Detection, Collection & Recovery Software

Design Framework
- Modular Design
  - Utilizes dynamic linking
  - Facilitates module update or repair
  - Efficient use of limited upload bandwidth
- Multitasking
  - Separate task for each module
  - Independent context eases error recovery
    - Kill task and restart [15]
- Task Synchronization and Communication
  - Operating system library primitives
Main Control Program
- Running the Computations
- Command Interpreter
  - Adding/deleting modules
  - Changing parameters
- Controlling the Watchdog Timers
- Logging Error
  - With or without collector module
- Error Recovery and Restart

Main Control Loop
```c
main()
{
    initialize();
    while (main_status == SHUTDOWN)
    {
        process_ground_command();
        run_next_test();
        send_heartbeat();
    }
}
```

Task Synchronization [15]
```c
run_test(int t)
{
    GiveSemaphore(TestStartSem[t]);
    result = TakeSemaphore(TestDoneSem, TimeoutAmount);
    if (result == ERROR)
        <delete TestProgram[t] and mark it as faulty>;
}
TestProgram_x()
{
    Initialize();
    FOREVER {
        TakeSemaphore(TestStartSem[x], WAIT_FOREVER);
        <do computation>
        GiveSemaphore(TestDoneSem);
    }
}
```

Watchdog Timers
- Watchdog Timer Module [13]
  - Limit total execution time of each test
  - Limit execution time of each step of a test
- Execution Completion Signal
  - Synchronized using semaphore
  - Limited wait time

Helper Modules
- Critical Modules
  - Periodically scrubbed for bit-flips in code

<table>
<thead>
<tr>
<th>Module</th>
<th>Error Detection</th>
<th>Scrubbing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Control</td>
<td>PP*, assertions</td>
<td>Periodic</td>
</tr>
<tr>
<td>Collector</td>
<td>PP, robust data structures</td>
<td>Periodic</td>
</tr>
<tr>
<td>Watchdog</td>
<td>PP, assertions</td>
<td>Periodic</td>
</tr>
<tr>
<td>Profiler</td>
<td>PP, assertions</td>
<td>Periodic</td>
</tr>
</tbody>
</table>

*PP: Programming Practices

Computation Modules
```plaintext
<table>
<thead>
<tr>
<th>Modules</th>
<th>Error Detection</th>
<th>Scrubbing</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT (4 variants)</td>
<td>ABFT</td>
<td>On-Demand</td>
</tr>
<tr>
<td></td>
<td>EDI</td>
<td>On-Demand</td>
</tr>
<tr>
<td></td>
<td>TMR</td>
<td>On-Demand</td>
</tr>
<tr>
<td></td>
<td>Mirror computation</td>
<td>On-Demand</td>
</tr>
<tr>
<td>Insert Sort</td>
<td>EDI (SSM+SAI)</td>
<td>On-Demand</td>
</tr>
<tr>
<td>LZW</td>
<td>EDI</td>
<td>On-Demand</td>
</tr>
<tr>
<td>Quick Sort</td>
<td>EDI</td>
<td>On-Demand</td>
</tr>
<tr>
<td>SortTMR</td>
<td>Diversified TMR</td>
<td>On-Demand</td>
</tr>
<tr>
<td>Unit targeting tests: ALU</td>
<td>N/A</td>
<td>Periodic</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td>Periodic</td>
</tr>
<tr>
<td>Other units</td>
<td></td>
<td>Periodic</td>
</tr>
<tr>
<td>Other Candidates</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>
```
### Error Distribution

- **Occurrence Probability**
  - Memory and CPU use
- **Computation Modules**
  - FFT, compression, sort, etc.
  - Large and time consuming
  - Detect error and abort
- **All Other Modules**
  - Small and quick
  - Not fully fault-tolerant

### Code Expansion

- **Limited Upload Bandwidth**
- **Small Code Size**
  - Low probability of error
- **Self-Expanding Program**
  - Run-time expansion
- **Error Rate**
  - CPU
  - Memory

\[
\lambda_1 = t_1 \lambda_{CPU} + s_1 \lambda_{memory} \\
\lambda_2 = t_2 \lambda_{CPU} + s_2 \lambda_{memory}
\]

### Error Detection Coverage

- **Detect Undetected Errors!**
  - Computation-based
  - e.g., sort
  - Fixed input
  - Check against known good result
- **Storing Known Good Results**
  - SEUs
  - Multiple copies
  - EDAC protection

### Error Handling and Logging

- **Main Control Module**
  - Quick handlers
  - Quick short report (error count)
- **Collector**
  - Complete log
  - Doubly-linked list
  - Error detection and correction
  - Scrubbing

### Error Recovery [15]

- **Continuous Error Collection**
  - Minimize need for ground control
  - Automatic recovery
- **Error in a Computation Module**
  - More probable errors
  - Disturbs the whole program
- **Critical Errors**
  - e.g., program hang-ups
  - Watchdog timers
- **Multitasking**
  - Kill task and restart

### Restart Steps

- **Error Detected by One of the Techniques**
- **Corresponding Handler Sets the Flags**
- **First Retry Attempted**
  - Pass: processor error
  - Fail: Request scrub and do second retry
- **Pass: memory bit-flip**
  - Fail: fatal error; need to reload module
ARGOS Experiment Software

Status
- Initial Test of Programs Completed
- Second Version of Software Currently Running
- Long Term Testing in Progress
  - Collect statistically significant num. of errors
  - Fine tune the software according to results

Data Analysis
- Error Rate
  - Processor activity
  - Orbital position
  - Sun activities
- Error Detection Techniques
  - Efficacy
  - Coverage vs. overhead tradeoffs
- Software-Implemented EDAC
  - Efficacy
  - Different codes and schemes

Difficulties
- Limited Communication Bandwidth
  - Object code < 40KB
  - Number of uploads
- Turn-Around Time
  - Output files
    - Delayed by 2-3 weeks
    - Out-of-order
  - Shut Downs

Future Work
- To-Do List
  - Code expansion
  - Self-checking/correction for EDAC module
  - Check against known good result
  - Profiler
- Future Research
  - More fault tolerance techniques
  - FPGA experiments

References (1)

References (2)