Resilience and Containment Domains

Sam Kaplan
ET International
January 21, 2015

Outline

- Background
- Status
- Code example
- Future Work

Background

- Exascale systems will have much higher rate of faults than current systems
  - Larger systems = more failures
  - Small components = more failures
  - More complex applications/data = more failures
- Traditional checkpointing schemes do not cope well with such high failure rates
  - Global checkpointing requires too much I/O
  - Local checkpointing may cause a “domino effect”
- Errors must be caught quickly to avoid propagation

Problem

- How can we allow application programmers to easily make their programs resilient to these faults?
- How can we reduce the overhead of traditional resilience (i.e. checkpointing)?
- How can we provide resilience in an extreme-scale, codelet-based environment?

Solution approach

- Containment Domains!
- Proposed by Mattan Erez and team at UT Austin
- http://lph.ece.utexas.edu/public/CDs/ContainmentDomains
- C++ API in development
  - Current implementation works only with serial programs

Containment domains

- Distributed, fine-grained, hierarchical method for error checking and recovery
- Allows optimization for specific applications or systems
- Maps well to codelet model
  - Each CD can be handled independently from any other CDs in the system
  - Only need to preserve data associated with currently active CD

Containment Domain API

- Preservation
  - Save input data for later recovery
- Body
  - Main algorithm function
- Detection
  - User-defined function to check correctness of results
  - Includes checks for hardware faults
  - If fault is detected, recover preserved data and re-run body

Containment Domain features

- Application is known to be in correct state before and after containment domain
  - Correct state is defined by detection function
- Tunable
  - Some CDs can be ignored if error rate is low enough
  - May regenerate input data algorithmically instead of storing
- Scalable
  - No coordination is needed
  - Multiple recoveries can occur simultaneously
  - Current model does not allow communication between CDs

Containment Domains in SWARM

- Basic feature set supported in SWARM prototype
  - Data preservation/recovery
  - User-defined detection functions
  - Continuation-based API to fit SWARM model
- No support for nested CDs (yet)
- Hardware failures can be simulated through random failures of a detection function

SWARM API

- swarm_ContainmentDomain_begin(THIS, begin, begin_ctxt, check, check_ctxt, done, done_ctxt)
  - begin: start of main body
  - check: checks for errors
    - On success, runs done()
    - On failure, re-runs begin()
  - done: cleanup and continue
- swarm_ContainmentDomain_preserve(THIS, data, length, id)
  - Save input data on first execution
  - Recover saved data on subsequent executions
  - Allows arbitrary number of preservations per CD
- swarm_ContainmentDomain_finish(THIS)
  - Close current CD and return to parent

SWARM API

- Next steps of CD are passed in NEXT and NEXT_THIS
  - NEXT and NEXT_THIS should be scheduled after body or check phase is complete
- Result of check function is passed as INPUT parameter
  - Depending on result of INPUT, either body or done codelet will be run next
- Able to re-run body and compare results without code duplication
- Everything else is handled internally in SWARM runtime

Code example

Codelet entry():

```
swarm_ContainmentDomain_init(cd);
/* set up contexts */
swarm_ContainmentDomain_begin(...);
```

Codelet begin():

```
swarm_ContainmentDomain_preserve(cd, &ctxt->A, sizeof(int), 0);
swarm_ContainmentDomain_preserve(cd, &ctxt->B, sizeof(int), 1);
*ctxt->C = ctxt->A * ctxt->B;
swarm_dispatch(NEXT, NEXT_THIS);
```

Code example

Codelet check():

```
int C2 = ctxt->A * ctxt->B;
success = (*ctxt->C == C2);
swarm_dispatch(NEXT, NEXT_THIS, success, NULL, NULL);
```

Codelet done():

```
swarm_ContainmentDomain_finish(cd);
swarm_shutdownRuntime(NULL);
```

Open questions

- What types of faults are expected?
  - Arithmetic errors
  - Memory errors
  - Node failure
- How can we recover from each type of error?
- How can these errors can be simulated on current hardware?
  - Randomly declare failure of check function, with certain probability
- What makes a good “check” function?
  - Checksum: quick but not complete
  - Run multiple times and compare results: thorough but slow

Future work

- Polishing prototype implementation
- Instrumenting a simple SWARM application with CDs
- Performance testing
  - Compare CD approach to checkpointing with various error rates