Containment Domains: a Full System Approach to Computational Resiliency

Containment Domains API (C++)

Revision 0.1

Michael Sullivan Ikhwan Lee Jinsuk Chung
Song Zhang Seong-Lyong Gong Derong Liu
Michael LeBeane Kyushick Lee Mattan Erez

Up to date HTML version at `lph.ece.utexas.edu/users/CDAPI`
`mailto:cds@lph.ece.utexas.edu`

Locality, Parallelism, and Hierarchy Group
Department of Electrical and Computer Engineering
The University of Texas at Austin

This research was, in part, funded by the U.S. Government with partial support from the Department of Energy under Awards DE-SC0008671 and DE-SC0008111. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

Generated by Doxygen 1.8.7 on Sun May 18 2014

Contents

0.1 Containment Domains 1

0.1.1 Containment Domains Overview 1

0.2 Examples 2

0.2.1 Examples 2

0.2.1.1 SpMV Example 2

0.3 Todo List 5

0.4 Module Index 5

0.4.1 Modules 5

0.5 Namespace Index 6

0.5.1 Namespace List 6

0.6 Class Index 6

0.6.1 Class List 6

0.7 File Index 6

0.7.1 File List 6

0.8 Module Documentation 7

0.8.1 CD-Related Definitions 7

0.8.1.1 Detailed Description 7

0.8.1.2 Enumeration Type Documentation 7

0.8.2 PGAS-Specific Methods and Types 9

0.8.2.1 Detailed Description 9

0.8.2.2 Enumeration Type Documentation 9

0.8.2.3 Function Documentation 9

0.8.3 Error Reporting 11

0.8.3.1 Detailed Description 11

0.8.3.2 Enumeration Type Documentation 11

0.8.3.3 Function Documentation 13

0.8.4 Preservation/Restoration Types and Methods 15

0.8.4.1 Detailed Description 15

0.8.4.2 Enumeration Type Documentation 15

0.8.4.3 Function Documentation 16
0.8.5 CD Init Functions 18
0.8.5.1 Detailed Description 18
0.8.5.2 Function Documentation 18
0.8.6 Global CD Accessor Functions 19
0.8.6.1 Detailed Description 19
0.8.6.2 Function Documentation 19
0.8.7 CD Hierarchy-Related Methods (create, begin, ...) 21
0.8.7.1 Detailed Description 21
0.8.7.2 Function Documentation 21
0.8.8 Detection and Recovery Methods 25
0.8.8.1 Detailed Description 25
0.8.8.2 Function Documentation 25
0.8.9 Methods for Interacting with the CD Framework and Tuner 29
0.8.9.1 Detailed Description 29
0.8.9.2 Function Documentation 29
0.8.10 Profiler-related methods 31
0.8.10.1 Detailed Description 31
0.8.10.2 Function Documentation 31
0.8.11 CD Event Functions for Non-Blocking Calls 32
0.8.11.1 Detailed Description 32
0.8.11.2 Function Documentation 32
0.8.12 Internal Functions for Customizable Recovery 33
0.8.12.1 Detailed Description 33
0.8.12.2 Function Documentation 33
0.9 Namespace Documentation 35
0.9.1 cd Namespace Reference 35
0.9.1.1 Detailed Description 36
0.10 Class Documentation 36
0.10.1 cd::CDEvent Class Reference 36
0.10.1.1 Detailed Description 37
0.10.1.2 Member Data Documentation 37
0.10.2 cd::CDHandle Class Reference 37
0.10.2.1 Detailed Description 38
0.10.2.2 Member Data Documentation 38
0.10.3 CDInternalPtr Class Reference 39
0.10.3.1 Detailed Description 39

0.10.4 cd::CDNameT Struct Reference 39
0.10.4.1 Detailed Description 39
0.10.5 cd::DegradedMemErrInfo Class Reference 40
0.10.5.1 Detailed Description 40
0.10.6 RecoverObject Class Reference 41
0.10.6.1 Detailed Description 41
0.10.6.2 Member Function Documentation 41
0.10.7 cd::RegenObject Class Reference 41
0.10.7.1 Detailed Description 42
0.10.7.2 Member Function Documentation 42
0.10.8 cd::SoftMemErrInfo Class Reference 42
0.10.8.1 Detailed Description 43
0.10.9 cd::SysErrT Struct Reference 43
0.10.9.1 Detailed Description 44
0.11 File Documentation 44
0.11.1 cd.h File Reference 44
0.11.1.1 Detailed Description 46
0.12 Example Documentation 46
0.12.1 spmv.cc 46
Index 50

0.1 Containment Domains

Containment Domains API v0.1 (C++)

Author

Kyushick Lee, Jinsuk Chung, Song Zhang, Seong-Lyong Gong, Derong Liu, Mattan Erez

Date

March 2014

Note

Print version available at http://lph.ece.utexas.edu/public/CDs

The purpose of this document is to describe a CD API for programming CD-enabled applications. A complete discussion of the semantics of containment domains is out of scope of this document; the latest version of the semantics is available at http://lph.ece.utexas.edu/public/CDs.

0.1.1 Containment Domains Overview

Containment domains (CDs) are a new approach that achieves low-overhead resilient and scalable execution (see http://lph.ece.utexas.edu/public/CDs). CDs abandon the prevailing one-size-fits-all approach to resilience and instead embrace the diversity of application needs, resilience mechanisms, and the deep hierarchies expected in exascale hardware and software. CDs give software a means to express resilience concerns intuitively and concisely. With CDs, software can preserve and restore state in an optimal way within the storage hierarchy and can efficiently support uncoordinated recovery. In addition, CDs allow software to tailor error detection, elision (ignoring some errors), and recovery mechanisms to algorithmic and system needs.

Todo Write a more significant introduction and put sections for what are currently modules

For now, the documentation is organized around the following "modules", which can also be accessed through the "Modules" tab on the HTML docs.

- CD Init Functions
- Global CD Accessor Functions
- CD Hierarchy-Related Methods (create, begin, ...)
- Preservation/Restoration Types and Methods
- Detection and Recovery Methods
  - Error Reporting
  - Internal Functions for Customizable Recovery
- Methods for Interacting with the CD Framework and Tuner
- PGAS-Specific Methods and Types
- CD Event Functions for Non-Blocking Calls
- CD-Related Definitions
- Examples

Note

This research was, in part, funded by the U.S. Government with partial support from the Department of Energy under Awards DE-SC0008671 and DE-SC0008111. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

0.2 Examples

0.2.1 Examples

0.2.1.1 SpMV Example

```
/*
Copyright 2014, The University of Texas at Austin
All rights reserved.

THIS FILE IS PART OF THE CONTAINMENT DOMAINS RUNTIME LIBRARY

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
  COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.
*/

/*
 * @file spmv.cc
 * @author Mattan Erez
 * @date March 2014
 *
 * @brief Hierarchical SpMV CD Example
 *
 * The SpMV computation consists of iteratively multiplying a constant
 * matrix by an input vector.  The resultant vector is then used as the
 * input for the next iteration.  We assume that the matrix and vector
 * are block partitioned and assigned to multiple nodes and cores.  This
 * simple application demonstrates many of the features of CDs and how
 * they can be used to express efficient resilience.
 *
 * One of the advantages of containment domains is that preservation and
 * recovery can be tailored to exploit natural redundancy within the
 * machine.  A CD does not need to fully preserve its inputs at the
 * domain boundary; partial preservation may be utilized to increase
 * efficiency if an input naturally resides in multiple locations.
 * Examples for optimizing preserve/restore/recover routines include
 * restoring data from sibling CDs or other nodes which already have a
 * copy of the data for algorithmic reasons.
 *
 * Hierarchical SpMV exhibits natural redundancy which can be
 * exploited through partial preservation and specialized
 * recovery. The input vector is distributed in such a way that
 * redundant copies of the vector are naturally distributed throughout
 * the machine. This is because there are \f$ N_0 \times N_0 \f$ fine-grained
 * sub-blocks of the matrix, but only \f$ N_0 \f$ sub-blocks in the vector.
 *
 * This is a hierarchical/recursive form of SpMV that uses some
 * pseudocode just to demonstrate the usage of the CD API
 */

#include "cd.h"

class VInRegen : public RegenType {
public:
  VInRegen(uint64_t task_num, uint subpartition) {
    task_num_ = task_num; subparition_ = subpartition;
  }

  CDErrType Regenerate(void* data_ptr, uint64_t len) {
    // During recovery (re-execution) Preserve acts like Restore
    // Writing this regeneration, which is really recovering data from
    // a sibling that has the same copy of the input vector is
    // difficult without assuming a specific parallelism
    // runtime. Unfortunately, both my MPI and UPC are rusty so I
    // can't do it right now. The idea is that we know which sibling
    // has data that we need based on the matrix partitioning and that
    // we know our subpartition number. We can then use the
    // parallelism runtime (or whoever tracked the task recursion) to
    // know which thread/rank/... this sibling is in, but we still
    // need to know its pointer to do a one-sided transfer of the data
    // because recovering this CD is independent of the sibling).
  }

protected:
  uint64_t task_num_;
  uint subpartition_;
};

/* @brief The recursive part that decomposes the problem for parallelism and containment.
 *
 * For simplicity, we assume that the input matrix has been
 * pre-partitioned to the appropriate number of levels to allow the
 * recursion to work correctly.
 *
 */
void SpMVRecurse(const SparseMatrix* matrix,
     const HierVector* v_in,
     HierVector*       v_out,
     const CDHandle* current_cd,
     uint  num_tasks
         ) {


  if (num_tasks > RECURSE_DEGREE) {
    uint tasks_per_child = num_tasks/RECURSE_DEGREE; // assume whole multiple
    for (int child=0; child < RECURSE_DEGREE; child++) {
      // assume that all iterations are all in parallel
      CDHandle* child_cd;
      // Creating the children CDs here so that we can more easily use
      // a collective mpi_comm_split-like Create method. This would be
      // easier if done internally by parallelism runtime/language
      child_cd = current_cd->CreateAndBegin(child, tasks_per_child);
      // Do some preservation
      CDEvent preserve_event;
      child_cd->Preserve(preserve_event,
       matrix->Subpartition(child), // Pointer to
       // start of subpartition within recursive matrix
        matrix->SubpartitionLen(child), // Length in
                // bytes of subpartition
        kCopy|kParent, // Can either create another
           // copy or use the parent's
           // preserved matrix with
           // appropriate offset
        "Matrix",
       "Matrix", matrix->PartitionOffset(),
       0,
       kReadOnly
       );
      // Regen object for input vector, assuming there is a
      // parallelism runtime that tracks recursion tree through task numbers
      VInRegen v_in_regen(ParRuntime::MyTaskNum(), child);
      child_cd->Preserve(preserve_event, // Chain this event
       v_in->Subpartition(child),
       v_in->SubpartitionLen(child),
       kCopy|kParent|kRegen,
       "vIn",
       "vIn", v_in->PartitionOffset(),
       &v_in_regen
       );
      // Do actual compute
      SpMVRecurse(matrix->Subpartition(child),
      v_in->Subpartition(child),
      v_out->Subpartition(child),
      child_cd, tasks_per_child);
      // Complete the CD
      preserve_event->Wait(); // Make sure preservation completed
      child_cd->Complete();
      child_cd->Destroy();
    }
  }
  else {
    for (int child=0; child < num_tasks; child++) {
      // assume that all iterations are all in parallel
      CDHandle* child_cd;
      CDHandle* child_cd = current_cd->Create();
      child_cd->Begin();
      // Do some preservation
      CDEvent preserve_event;
      child_cd->Preserve(preserve_event,
       matrix->Partition(), // Pointer to
       // start of subpartition within recursive matrix
       matrix->PartitionLen(), // Length in
       // bytes of subpartition
       kCopy|kParent, // Can either create another
       // copy or use the parent's
       // preserved matrix with
       // appropriate offset
       "Matrix",
       "Matrix", matrix->PartitionOffset(),
       0,
       kReadOnly
       );
      VInRegen v_in_regen(ParRuntime::MyTaskNum(), child);
      child_cd->Preserve(preserve_event, // Chain this event
       v_in->Partition(),
       v_in->SubpartitionLen(),
       kCopy|kParent|kRegen,
       "vIn",
       "vIn", v_in->PartitionOffset(),
       &v_in_regen
       );
      // Do actual compute
      SpMVLeaf(matrix->Subpartition(child),
         v_in->Subpartition(child),
         v_out->Subpartition(child),
         child_cd, tasks_per_child);
      child_cd->Complete();
      child_cd->Destroy();
    }
  }

  v_out->ReduceSubpartitions(num_tasks);
}

void SpMVLeaf(const SparseMatrix* matrix,
        const HierVector* v_in,
        HierVector*       v_out,
        const CDHandle* current_cd,
        uint  num_tasks
        ) {
  for (uint row=0; row < matrix->NumRows(); row++) {
    v_out[row] = 0.0;
    for (unit col = matrix->RowStart[row];
   col <  matrix->RowStart[row+1];
   col++) {
      uint prev_idx = 0;
      uint idx = matrix->Index[col];
      v_out[row] += matrix->NonZero[col]*v_in[idx];
      CDAssert(idx >= prev_idx); // data structure sanity check
      prev_idx = idx;
    }
  }
}
```

0.3 Todo List

Member cd::CDHandle::GetErrorProbability (SysErrT error_type, uint error_num,)

Decide on rate vs. number+probability

Member cd::CDHandle::RegisterRecovery (uint error_name_mask, uint error_loc_mask, RecoverObject ∗recover_object=0)

Does registering recovery also imply turning on detection? Or is that done purely through RequireErrorProbability()?

Member cd::CDHandle::RequireErrorProbability (SysErrT error_type, uint error_num, float probability, bool fail_over=true)

Decide on rate vs. number+probability

Member cd::CDHandle::SetPGASUsage (void ∗data_ptr, uint64_t len, PGASUsageT region_type=kShared)

Do we want to expose explicit logging functions?

Class cd::CDNameT

Decide on whether to represent a CD name as (level, num) or as `typedef std::vector<uint> CDName↩ T;` to represent the path through the tree branches.

Member cd::SysErrLocT

is SysErrLocT comprehensive enough for portability?

Member cd::SysErrNameT

Is SysErrNameT comprehensive enough for portability?

segv (segmentation violations) can be used as proxy for soft memory errors using the existing kernel infrastructure

Group cd_error_probability

What about specifying leniant communication-related errors for relaxed-CDs context?

Member CDInternalPtr::InternalReexecute ()

Discuss other aspects of reexecution (e.g., logging). [FIXME] Discuss other aspects of reexecution (e.g., logging).

page Containment Domains

Write a more significant introduction and put sections for what are currently modules

Class RecoverObject

Write some example for custom recovery (see GVR interpolation example, although they do it between versions).

0.4 Module Index

0.4.1 Modules

Here is a list of all modules:

CD-Related Definitions 7
PGAS-Specific Methods and Types 9
Error Reporting 11
Preservation/Restoration Types and Methods 15
CD Init Functions 18
Global CD Accessor Functions 19
CD Hierarchy-Related Methods (create, begin, ...) 21

Detection and Recovery Methods 25
Methods for Interacting with the CD Framework and Tuner 29
Profiler-related methods 31
CD Event Functions for Non-Blocking Calls 32
Internal Functions for Customizable Recovery 33

0.5 Namespace Index

0.5.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

cd
Containment Domains namespace for global functions, types, and main interface 35

0.6 Class Index

0.6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

cd::CDEvent
An object that provides an event identifier to a non-blocking CD runtime call 36
cd::CDHandle
An object that provides a handle to a specific CD instance 37
CDInternalPtr
A class that represents the interface to the internal implementation of an actual CD 39
cd::CDNameT
A type to uniquely name a CD in the tree 39
cd::DegradedMemErrInfo
Interface to degraded memory error information 40
RecoverObject
Recovery method that can be inherited and specialized by user 41
cd::RegenObject
Interface for specifying regeneration functions for preserve/restore 41
cd::SoftMemErrInfo
Interface to soft memory error information 42
cd::SysErrT
Type for specifying errors and failure 43

0.7 File Index

0.7.1 File List

Here is a list of all documented files with brief descriptions:

cd.h
Containment Domains API v0.1 (C++) 44

0.8 Module Documentation

0.8.1 CD-Related Definitions

Classes

- struct cd::CDNameT

  A type to uniquely name a CD in the tree.

Enumerations

- enum cd::CDModeT { cd::kStrict =0, cd::kRelaxed }

  Type for specifying whether a CD is strict or relaxed.
- enum cd::CDExecutionModeT { cd::kExec =0, cd::kReexec }

  Type for specifying whether the current CD is executing for the first time or is currently reexecuting as part of recovery.
- enum cd::PreserveUseT { cd::kUnsure =0, cd::kReadOnly = 1, cd::kReadWrite = 2 }

  Type to indicate whether preserved data is from read-only or potentially read/write application data.

0.8.1.1 Detailed Description

The CD-Related Definitions module includes general CD-related type definitions that don't fall into other type definitions (Error Reporting and Preservation/Restoration Types and Methods).

0.8.1.2 Enumeration Type Documentation

0.8.1.2.1 enum cd::CDExecutionModeT

Type for specifying whether the current CD is executing for the first time or is currently reexecuting as part of recovery.

During reexecution, data is restored instead of being preserved. Additionally, for relaxed CDs, some communication and synchronization may not repeated and instead preserved (logged) values are used. See `http://lph.ece.↩ utexas.edu/public/CDs` for a detailed description. Note that this is not part of the cd_internal namespace because the application programmer may want to manipulate this when specifying specialized recovery routines.

Enumerator

kExec First execution.

kReexec Rexecution.

0.8.1.2.2 enum cd::CDModeT

Type for specifying whether a CD is strict or relaxed.

This type is used to specify whether a CD is strict or relaxed. The full definition of the semantics of strict and relaxed CDs can be found in the semantics document under `http://lph.ece.utexas.edu/public/CDs`. In brief, concurrent tasks (threads, MPI ranks, ...) in two different strict CDs cannot communicate with one another and must first complete inner CDs so that communicating tasks are at the same CD context. Tasks in two different relaxed CDs may communicate (verified data only). Relaxed CDs typically incur additional runtime overhead compared to strict CDs.

Enumerator

kStrict A strict CD.

kRelaxed A relaxed CD.

0.8.1.2.3 enum cd::PreserveUseT

Type to indicate whether preserved data is from read-only or potentially read/write application data.

See also

CDHandle::Preserve(), CDHandle::Complete()

Enumerator

kUnsure by the CD (treated as Read/Write for now, but may be optimized later) Not sure whether data being preserved will be written

kReadOnly Data to be preserved is read-only within this CD.

kReadWrite Data to be preserved will be modified by this CD.

0.8.2 PGAS-Specific Methods and Types

Enumerations

- enum cd::PGASUsageT { cd::kShared = 0, cd::kPotentiallyShared, cd::kPrivatized, cd::kPrivate }

  Different types of PGAS memory behavior for relaxed CDs.

Functions

- CDErrT cd::CDHandle::SetPGASUsage (void ∗data_ptr, uint64_t len, PGASUsageT region_type=kShared)

  Declare how a region of memory behaves within this CD (for Relaxed CDs)
- CDErrT cd::CDHandle::SetPGASOwnerWrites (void ∗data_ptruint64_t len, bool owner_writes=true)

  Simplify optimization of discarding relaxed CD log entries.

0.8.2.1 Detailed Description

0.8.2.2 Enumeration Type Documentation

0.8.2.2.1 enum **cd::PGASUsageT**

Different types of PGAS memory behavior for relaxed CDs.

Please see the CD semantics document at `http://lph.ece.utexas.edu/public/CDs` for a full description of PGAS semantics. In brief, because of the logging/tracking requirements of relaxed CDs, it is important to identify which memory accesses may be for communication between relaxed CDs vs. memory accesses that are private (or temporarily privatized) within this CD.

Enumerator

kShared Definitely shared for actual communication.

kPotentiallyShared CD, essentially equivalent to kShared for CDs. Perhaps used for communication by this

kPrivatized communication during this CD. Shared in general, but not used for any

kPrivate Entirely private to this CD.

0.8.2.3 Function Documentation

0.8.2.3.1 CDErrT cd::CDHandle::SetPGASOwnerWrites (void ∗data_ptruint64_t *len,* bool *owner_writes* = `true`)

Simplify optimization of discarding relaxed CD log entries.

When using relaxed CDs, the CD runtime may log all communication with tasks that are in a different CD context. While privatizing some accesses reduces logging volume, all logged entries must still be propagated and preserved up the CD tree for distributed recovery. Log entries may be discarded when both the task that produced the data and the task that logged it are in the same CD (after descendant CDs complete and "merge"). This can always be guaranteed when the least-common strict ancestor is reached, but may happen sooner. In order to identify the earliest opportunity to discard a log entry, the CD runtime must track producers, which is impractical in general. In the specific and common scenario of "owner computes", however, it is possible to track the producer with low overhead. The SetPGASOwnerWrites() method i used to indicate this behavior.

Returns

kOK on success.

Parameters

`in`	*len*	pointer to data to be "Typed"; __currently must be in same address space as calling task, but will extend to PGAS fat pointers later [in] Length of preserved data (Bytes)
`in`	*owner_writes*	Is the current CD the only CD in which this address range is written (until strict ancestor is reached)?

0.8.2.3.2 CDErrT cd::CDHandle::SetPGASUsage (void $*$ *data_ptr*, uint64_t *len*, PGASUsageT *region_type* = kShared)

Declare how a region of memory behaves within this CD (for Relaxed CDs)

Declare the behavior of a region of PGAS/GAS memory within this CD to minimize logging/tracking overhead. Ideally, only those memory accesses that really are used to communicate between this relaxed CD and another relaxed CD are logged/tracked.

Warning

For now, we are using an address to mark the type, but it is quite possible that using actual types and casting is better. Unfortunately types cannot be done through an API interface and require a change to the language. It is not clear how much overhead will be saved through just this API technique and we will explore language changes (see discussion below).

In the UPC++ implementation, this API call should not be used, and a cast from `shared_array` to `privatized←_array` (or `shared_var` to `privatized_var`) is preferred. UPC++ implementation should be quite straightforward

Todo Do we want to expose explicit logging functions?

Discussion on 3/11/2014:

Just in UPC runtime, perhaps cram a field that says log vs. unlogged into the pointer representation (steal one bit). That we can perhaps do just in the runtime. A problem is that all pointer operations (e.g., comparisons) need to know about this bit.

If adding to the compiler, then should be done at same point as strict and relaxed are done.

There is also a pragma that can be used for changing the default behavior from shared to privatized (assuming that all or at least vast majority of accesses) within a code block are such. This might be easier than casting.

Parameters

`in`	*data_ptr*	pointer to data to be "Typed"; __currently must be in same address space as calling task, but will extend to PGAS fat pointers later
`in`	*len*	Length of preserved data (Bytes)
`in`	*region_type*	How is this memory range used (shared for comm or not?)

0.8.3 Error Reporting

Classes

- class cd::SoftMemErrInfo

  Interface to soft memory error information.
- class cd::DegradedMemErrInfo

  Interface to degraded memory error information.
- struct cd::SysErrT

  Type for specifying errors and failure.

Enumerations

- enum cd::SysErrNameT {
  cd::kOK =0, cd::kSoftMem = 0b1, cd::kDegradedMem = 0b01, cd::kSoftComm = 0b001,
  cd::kDegradedComm = 0b0001, cd::kSoftComp = 0b00001, cd::kDegradedResource =0b000001, cd::kHard←Resource = 0b0000001,
  cd::kFileSys = 0b00000001 }

  Type for specifying system errors and failure names.
- enum cd::SysErrLocT {
  cd::kOK =0, cd::kIntraCore = 0b1, cd::kCore = 0b01, cd::kProc = 0b001,
  cd::kNode = 0b0001, cd::kModule = 0b00001, cd::kCabinet = 0b000001, cd::kCabinetGroup =0b0000001,
  cd::kSystem = 0b00000001 }

  Type for specifying errors and failure location names.
- enum cd::CDErrT { cd::kOK =0, cd::kAlreadyInit, cd::kError }

  Type for specifying error return codes from an API call – signifies some failure of the API call itself, not a system failure.

Functions

- uint cd::DeclareErrName (const char ∗name_string)

  Create a new error/failure type name.
- CDErrT cd::UndeclareErrName (uint error_name_id)

  Free a name that was created with DeclareErrorName()
- uint cd::DeclareErrLoc (const char ∗name_string)

  Create a new error/failure type name.
- CDErrT cd::UndeclareErrLoc (uint error_name_id) class SysErrInfo

  Free a name that was created with DeclareErrLoc()

0.8.3.1 Detailed Description

The Error Reporting module includes the definition of types and methods used for system and CD runtime error/failure reporting.

0.8.3.2 Enumeration Type Documentation

0.8.3.2.1 enum cd::CDErrT

Type for specifying error return codes from an API call – signifies some failure of the API call itself, not a system failure.

Unlike SysErrNameT, CDErrT is not for system errors, but rather errors originating from the CD framework itself.

For now, only returning OK or error, but will get more elaborate in future versions of this API.

Enumerator

kOK No errors/failures. Call executed without error.

kAlreadyInit Init called more than once.

kError Call did not execute as expected.

0.8.3.2.2 enum cd::SysErrLocT

Type for specifying errors and failure location names.

Please see SysErrNameT for discussion of intent and defintions

This is really not that suitable for all topologies, but the intent really is to be rather comprehensive to maintain portability – how do we resolve this?

Todo is SysErrLocT comprehensive enough for portability?

See also

SysErrNameT, DeclareErrLoc(), UndeclareErrLoc()

Enumerator

kOK No errors/failures. Call executed without error.

kIntraCore Within a part of a core.

kCore A core.

kProc Processor.

kNode Same as processor?

kModule Module.

kCabinet A cabinet.

kCabinetGroup Some grouping of cabinets.

kSystem Entire system.

0.8.3.2.3 enum cd::SysErrNameT

Type for specifying system errors and failure names.

This type represents the interface between the user and the system with respect to errors and failures. The intent is for these error/failure names to be fairly comprehensive with respect to system-based issues, while still providing general/abstract enough names to be useful to the application programmer. The use categories/names are meant to capture recovery strategies that might be different based on the error/failure and be comprehensive in that regard. The DeclareErrName() method may be used to create a programmer-defined error that may be associated with a programmer-provided detection method.

We considered doing an extensible class hierarchy like GVR, but ended up with a hybrid type system. There are predefined bit vector constants for error/failure names and machine location names. These may be extended by application programmers for specialized detectors. These types are meant to capture abstract general classes of errors that may be treated differently by recovery functions and therefore benefit from easy-to-access and well-defined names. Additional error/failure-specific information will be represented by the SysErrInfo interface class hierarchy, which may be extended by the programmer at compiler time. Thus, each error/failure is a combination of SysErrNameT, SysErrLocT, and SysErrInfo.

This needs more thought

Warning

> The SysErrNameT and SysErrLocT are extensible by a runtime call to generate a new constant, but SysErrInfo is a class hierarchy and extended at compile time by inhereting the interface – is this a problem? Should we go the GVR way with all extensions done at runtime and all accesses with potential runtime methods and no compile-time typing?

Todo Is SysErrNameT comprehensive enough for portability?

segv (segmentation violations) can be used as proxy for soft memory errors using the existing kernel infrastructure

See also

> SysErrLocT, DeclareErrName(), UndeclareErrName()

Enumerator

- ***kOK*** No errors/failures. Call executed without error.
- ***kSoftMem*** Soft memory error (info includes address range and perhaps syndrome)
- ***kDegradedMem*** Hard memory error that disabled some memory capacity (info includes address range(s))
- ***kSoftComm*** (info includes message info) Soft communication error
- ***kDegradedComm*** Some channel loss.
- ***kSoftComp*** includes affected PC and perhaps bounds on the error?) Soft compute error (info
- ***kDegradedResource*** functionality Resource lost **some**
- ***kHardResource*** (control/reachability failure). Resource entirely lost
- ***kFileSys*** Some file

0.8.3.3 Function Documentation

0.8.3.3.1 uint cd::DeclareErrLoc (const char ∗ *name_string*)

Create a new error/failure type name.

Returns

> Returns a "constant" corresponding to a free bit location in the SysErrNameT bitvector.

See also

> SysErrNameT, SysErrLocT, UndeclareErrLoc()

Parameters

name_string	user-specified name for a new error/failure location

0.8.3.3.2 uint cd::DeclareErrName (const char ∗ *name_string*)

Create a new error/failure type name.

Returns

> Returns a "constant" corresponding to a free bit location in the SysErrNameT bitvector.

See also

> SysErrNameT, SysErrLocT, UndeclareErrName()

Parameters

name_string	user-specified name for a new error/failure type

0.8.3.3.3 CDErrT cd::UndeclareErrLoc (uint *error_name_id*)

Free a name that was created with DeclareErrLoc()

Returns

Returns kOK on success.

See also

SysErrNameT, SysErrLocT, DeclareErrLoc()Interface to error/failure-specific information

An abstract interface to specific error/failure information, such as address range, core number, degradation, specific lost functionality, ...

This is an empty interface because the information is very much error dependent. Also defining a few specific initial examples below. This follows the GVR ideas pretty closely.

See also

SoftMemErrInfo, DegradedMemErrInfo

Parameters

error_name_id	ID to free

0.8.3.3.4 CDErrT cd::UndeclareErrName (uint *error_name_id*)

Free a name that was created with DeclareErrorName()

Returns

Returns kOK on success.

See also

SysErrNameT, SysErrLocT, DeclareErrName()

Parameters

error_name_id	ID to free

0.8.4 Preservation/Restoration Types and Methods

Classes

- class cd::RegenObject

  Interface for specifying regeneration functions for preserve/restore.

Enumerations

- enum cd::PreserveMechanismT { cd::kCopy =0b001, cd::kRef =0b010, cd::kRegen =0b100 }

  Type for specifying preservation methods.

Functions

- CDErrT cd::CDHandle::Preserve (void ∗data_ptr, uint64_t len, uint_t preserve_mask=kCopy, const char ∗my↩ _name=0, const char ∗ref_name=0, uint_64t ref_offset=0, const RegenObject ∗regen_object=0, PreserveUseT data_usage=kUnsure)

  Preserve data to be restored when recovering (typically reexecuting the CD from right after its Begin() call.
- CDErrT cd::CDHandle::Preserve (CDEvent &cd_event, void ∗data_ptr, uint64_t len, uint_t preserve_mask=kCopy, const char ∗my_name=0, const char ∗ref_name=0, uint_64t ref_offset=0, const RegenObject ∗regen_object=0, PreserveUseT data_usage=kUnsure)

  Non-blocking preserve data to be restored when recovering (typically reexecuting the CD from right after its Begin() call.

0.8.4.1 Detailed Description

The Preservation/Restoration Types and Methods module contains all preservation/restoration related types and methods.

0.8.4.2 Enumeration Type Documentation

0.8.4.2.1 enum **cd::PreserveMechanismT**

Type for specifying preservation methods.

See `http://lph.ece.utexas.edu/public/CDs` for a detailed description.

The intent is for this to be used as a mask for specifying multiple legal preservation methods so that the autotuner can choose the appropriate one.

See also

RegenObject, CDHandle::Preserve()

Enumerator

kCopy Prevervation via copy copies the data to be preserved into another storage/mem location

kRef Preservation via reference indicates that restoration can occur by restoring data that is already preserved in another CD. **Restriction:** in the current version of the API only the parent can be used as a reference.

kRegen Preservation via regenaration is done by calling a user-provided function to regenerate the data during restoration instead of copying it from preserved storage.

0.8.4.3 Function Documentation

0.8.4.3.1 CDErrT cd::CDHandle::Preserve (void ∗ *data_ptr*, uint64_t *len*, uint_t *preserve_mask* = **kCopy**, const char ∗ *my_name* = `0`, const char ∗ *ref_name* = `0`, uint_64t *ref_offset* = `0`, const **RegenObject** ∗ *regen_object* = `0`, **PreserveUseT** *data_usage* = **kUnsure**)

Preserve data to be restored when recovering (typically reexecuting the CD from right after its Begin() call.

Preserves application data so that it can be restored for correct CD recovery (typically reexecution).

Preserve() preserves data on the first execution of the CD (`kExec`) but acts to restore data when a CD is reexecuted (`kReexec`). Success is defined as successfully preserving or restoring `len` bytes of contiguous data starting at address `data_ptr`.

In many cases there is more than one way to preserve data and the best way to do depends on machine-specific characteristics. It is therefore possible to call Preserve() with a set of possible correct and legal preservation methods and have an autotuner select the best one. This is done by setting the appropriate bits in the `preserve_mask` parameter based on the constants defined by PreserveMethodT.

Returns

kOK on success and kError otherwise.

See also

Complete()

Parameters

`in`	*data_ptr*	pointer to data to be preserved; __currently must be in same address space as calling task, but will extend to PGAS fat pointers later
`in`	*len*	Length of preserved data (Bytes)
`in`	*preserve_mask*	Allowed types of preservation (e.g., kCopy | kRegen), default only via copy
`in`	*my_name*	Optional C-string representing the name of this preserved data for later preserve-via-reference
`in`	*ref_name*	Optional C-string representing a user-specified name that was set by a previous preserve call at the parent.; **Do we also need an offset into parent preservation?**
`in`	*ref_offset*	explicit offset
	regen_object	within the named region at the other CD (for restoration via reference) [in] optional user-specified function for regenerating values instead of restoring by copying
`in`	*data_usage*	This flag is used to optimize consecutive Complete/Begin calls where there is significant overlap in preserved state that is unmodified (see Complete()).

0.8.4.3.2 CDErrT cd::CDHandle::Preserve (**CDEvent** & *cd_event*, void ∗ *data_ptr*, uint64_t *len*, uint_t *preserve_mask* = **kCopy**, const char ∗ *my_name* = `0`, const char ∗ *ref_name* = `0`, uint_64t *ref_offset* = `0`, const **RegenObject** ∗ *regen_object* = `0`, **PreserveUseT** *data_usage* = **kUnsure**)

Non-blocking preserve data to be restored when recovering (typically reexecuting the CD from right after its Begin() call.

Preserves application data so that it can be restored for correct CD recovery (typically reexecution).

Preserve() preserves data on the first execution of the CD (`kExec`) but acts to restore data when a CD is reexecuted (`kReexec`). Success is defined as successfully preserving or restoring `len` bytes of contiguous data starting at address `data_ptr`.

In many cases there is more than one way to preserve data and the best way to do depends on machine-specific characteristics. It is therefore possible to call Preserve() with a set of possible correct and legal preservation methods and have an autotuner select the best one. This is done by setting the appropriate bits in the `preserve_mask` parameter based on the constants defined by PreserveMethodT.

The CDEvent object will be initialized to wait on this particular call if it is empty. If the CDEvent already contains an event, then this new event is chained to it – in this way, the user can use a single CDEvent::Wait() call to wait on a sequence of non-blocking calls.

Returns

kOK on success and kError otherwise.

See also

Complete()

Parameters

`in,out`	*cd_event*	enqueue this call onto the cd_event
`in`	*data_ptr*	pointer to data to be preserved; __currently must be in same address space as calling task, but will extend to PGAS fat pointers later
`in`	*len*	Length of preserved data (Bytes)
`in`	*preserve_mask*	Allowed types of preservation (e.g., kCopy | kRegen), default only via copy
`in`	*my_name*	Optional C-string representing the name of this preserved data for later preserve-via-reference
`in`	*ref_name*	Optional C-string representing a user-specified name that was set by a previous preserve call at the parent.; **Do we also need an offset into parent preservation?**
`in`	*ref_offset*	explicit offset
	regen_object	within the named region at the other CD [in] optional user-specified function for regenerating values instead of restoring by copying
`in`	*data_usage*	This flag is used to optimize consecutive Complete/Begin calls where there is significant overlap in preserved state that is unmodified (see Complete()).

0.8.5 CD Init Functions

Functions

- CDHandle ∗ cd::Init (bool collective=trueCDErrT ∗error=0)

  Initialize the CD runtime.

0.8.5.1 Detailed Description

The CD Init Functions are used for initialization.

0.8.5.2 Function Documentation

0.8.5.2.1 CDHandle∗ cd::Init (bool *collective* = `trueCDErrT *error=0`)

Initialize the CD runtime.

Creates all necessary CD runtime components and data structures, initializes the CD runtime, and creates and begins the root CD. At this point, the current CD is the root. `cd::Init()` **should only be called once per application.**

There are two variants of this function. The first is a collective operation across all SPMD tasks currently in the application. All tasks get a handle to the single root and begin the CD in a synchronized manner. The second variant is called locally by a single task and synchronization, as well as broadcasting the handle are up to the programmer. Use of this second variant is discouraged.

Returns

Returns a handle to the root CD; Returns `kOK` if successful, `AlreadyInit` if called more than once, and `k←Error` if initialization is unsuccessful for some reason.

The handle to the root is also registered in some globally accessible variable so that it can be accessed by cd::Get←CurrentCD() and cd::GetRootCD().

Parameters

`in`	*collective*	Collective operation (default) or called from only a single task (discouraged) [in,out] Pointer for error return value (kOK on success, kAlreadyInit if trying to re-initialize, and kError on other failures); no error value returned if error=0.

0.8.6 Global CD Accessor Functions

Functions

- CDHandle ∗ cd::GetRootCD ()

  Accessor function to root CD of the application.
- CDHandle ∗ cd::GetCurrentCD ()

  Accessor function to current active CD.
- CDErrT cd::SetCurrentCD (const CDHandle ∗cd)

  Accessor function for setting the current active CD.

0.8.6.1 Detailed Description

The Global CD Accessor Functions are used to get the current and root CD handles if these are not explicitly tracked.

These methods are globally accessible without a CDHandle object.

0.8.6.2 Function Documentation

0.8.6.2.1 CDHandle∗ cd::GetCurrentCD ( )

Accessor function to current active CD.

At any point after the CD runtime is initialized, each task is associated with a current CD instance. The current CD is the deepest CD in the tree visible from the task that has begun but has not yet completed. In other words, whenever a CD begins, it becomes the current CD. When a CD completes, its parent becomes the current CD.

Returns

returns a pointer to the handle of the current active CD; Returns 0 if CD runtime is not yet initialized or because of a CD implementation bug.

0.8.6.2.2 CDHandle∗ cd::GetRootCD ( )

Accessor function to root CD of the application.

Returns

returns a pointer to the handle of the root CD; Returns 0 if CD runtime is not yet initialized or because of a CD implementation bug.

0.8.6.2.3 CDErrT cd::SetCurrentCD (const CDHandle ∗ *cd*)

Accessor function for setting the current active CD.

At any point after the CD runtime is initialized, each task is associated with a current CD instance. The current CD is the deepest CD in the tree visible from the task that has begun but has not yet completed. In other words, whenever a CD begins, it becomes the current CD. When a CD completes, its parent becomes the current CD.

This function is needed when using a non-collective CDHandle::Begin() or CDHandle::Complete() and the CD that is now beginning contains multiple tasks.

Returns

returns a pointer to the handle of the current active CD; Returns 0 if CD runtime is not yet initialized or because of a CD implementation bug.

See also

CDHandle::Begin(), CDHandle::Complete()

Parameters

`in`	*cd*	pointer to CDHandle of the CD instance that is now the current CD.

0.8.7 CD Hierarchy-Related Methods (create, begin, ...)

Functions

- CDHandle ∗ cd::CDHandle::Create (char ∗name=0, CDModeT type=kStrict, uint error_name_mask=0, uint error_loc_mask=0, CDErrT ∗error=0)

  Single-task non-collective Create.
- CDHandle ∗ cd::CDHandle::Create (uint_t color, uint_t num_tasks_in_color, char ∗name=0, CDModeT type=k↩Strict, uint error_name_mask=0, uint error_loc_mask=0, CDErrT ∗error=0)

  Collective Create.
- CDHandle ∗ cd::CDHandle::CreateAndBegin (uint_t color, uint_t num_tasks_in_color, char ∗name=0, CDModeT type=kStrict, uint error_name_mask=0, uint error_loc_mask=0, CDErrT ∗error=0)

  Collective Create+Begin.
- CDErrT cd::CDHandle::Destroy (bool collective=false)

  Destroys a CD.
- CDErrT cd::CDHandle::Begin (bool collective=true)

  Begins a CD.
- CDErrT cd::CDHandle::Complete (bool collective=true, bool update_preservations,)

  Completes a CD.
- CDNameT cd::CDHandle::GetName ()

  Get the name/location of this CD.
- CDHandle ∗ cd::CDHandle::GetParent ()

  Get CDHandle to this CD's parent.

0.8.7.1 Detailed Description

0.8.7.2 Function Documentation

0.8.7.2.1 CDErrT cd::CDHandle::Begin (bool *collective* = `true`)

Begins a CD.

Begins the CD and sets it as the current CD for the calling task. If `collective=true` then the Begin() call is a collective across all tasks that collectively created this CD. It is illegal to call a collective Begin() on a CD that was created without a collective Create(). If `collective=false`, the user is responsible for first synchronizing all tasks contained within this CD and then updating the current CD in each task using cd::SetCurrentCD().

Important constraint: Begin() and Complete() must be called from within the same program scope (i.e., same degree of scope nesting).

Returns

Returns kOK when successful and kError otherwise.

See also

Complete()

Parameters

`in`	*collective*	Specifies whether this call is a collective across all tasks contained by this CD or whether its to be run by a single task only with the programmer responsible for synchronization.

0.8.7.2.2 CDErrT cd::CDHandle::Complete (bool *collective* = `true`, bool *update_preservations*)

Completes a CD.

Completes the CD and sets its parent as the current CD for the calling task. If `collective=true` then the Complete() call is a collective across all tasks that collectively created this CD. It is illegal to call a collective Complete() on a CD that was created without a collective Create(). If `collective=false`, the user is responsible for first synchronizing all tasks contained within this CD and then updating the current CD in each task using cd::SetCurrentCD().

Important constraint: Begin() and Complete() must be called from within the same program scope (i.e., same degree of scope nesting).

Warning

The update preservation (advance) optimization semantics may not yet be supported.

Returns

Returns kOK when successful and kError otherwise.

See also

Begin()

Parameters

`in`	*collective*	Specifies whether this call is a collective across all tasks contained by this CD or whether its to be run by a single task only with the programmer responsible for synchronization
`in`	*update_↩ preservations*	Flag that indicates whether preservation should be updated on Complete rather than discarding all preserved state. If `true` then Complete followed by Begin can be much more efficient if the majority of preserved data overlaps between these two consecutive uses of the CD object (this enables the Cray CD AdvancePoint↩ InTime functionality).

0.8.7.2.3 CDHandle∗ cd::CDHandle::Create (char ∗ *name* = `0`, **CDModeT** *type* = **kStrict**, uint *error_name_mask* = `0`, uint *error_loc_mask* = `0`, **CDErrT** ∗ *error* = `0`)

Single-task non-collective Create.

Creates a new CD as a child of this CD. The new CD does not begin until Begin() is called explicitly.

This version of Create() is intended to be called by only a single task and the value of the returned handle explicitly communicated between all tasks contained within the new child. An alternate collective version is also provided. It is expected that this non-collective version will be mostly used within a single task or, at least, within a single process address space.

Returns

Returns a pointer to the handle of the newly created child CD; returns 0 on an error (error code returned in a parameter).

Parameters

`in`	*name*	Optional user-specified name that can be used to "re-create" the same CD object if it was not destroyed yet; useful for resuing preserved state in CD trees that are not loop based.
`in`	*type*	Strict or relaxed
`in`	*error_name_↩ mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in`	*error_loc_mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in,out`	*error*	Pointer for error return value (kOK on success and kError on failure); no error value returned if error=0.

0.8.7.2.4 CDHandle∗ cd::CDHandle::Create (uint_t *color,* uint_t *num_tasks_in_color,* char ∗ *name* = `0`, CDModeT *type* = kStrict, uint *error_name_mask* = `0`, uint *error_loc_mask* = `0`, CDErrT ∗ *error* = `0`)

Collective Create.

Creates a new CD as a child of the current CD. The new CD does not begin until CDHandle::Begin() is called explicitly.

This version of Create() is intended to be called by all tasks that will be contained in the new child CD. It functions as a collective operation in a way that is analogous to MPI_comm_split, but only those tasks that are contained in the new child synchronize with one another.

Returns

Returns a pointer to the handle of the newly created child CD; returns 0 on an error.

Parameters

`in`	*color*	The "color" of the new child to which this task will belong
`in`	*num_tasks_in_↩ color*	The total number of tasks that are collectively creating the child numbered "color"; the collective waits for this number of tasks to arrive before creating the child
`in`	*name*	Optional user-specified name that can be used to "re-create" the same CD object if it was not destroyed yet; useful for resuing preserved state in CD trees that are not loop based.
`in`	*type*	Strict or relaxed
`in`	*error_name_↩ mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in`	*error_loc_mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in,out`	*error*	Pointer for error return value (kOK on success and kError on failure); no error value returned if error=0.

0.8.7.2.5 CDHandle∗ cd::CDHandle::CreateAndBegin (uint_t *color,* uint_t *num_tasks_in_color,* char ∗ *name* = `0`, CDModeT *type* = kStrict, uint *error_name_mask* = `0`, uint *error_loc_mask* = `0`, CDErrT ∗ *error* = `0`)

Collective Create+Begin.

Creates a new CD as a child of the current CD. The new CD then immediately begins with a single collective call.

This version of is intended to be called by all tasks that will be contained in the new child CD. It functions as a collective operation in a way that is analogous to MPI_comm_split, but only those tasks that are contained in the new child synchronize with one another. To avoid unnecessary collectives, CreateAndBegin() then immediately begins the new CD.

Returns

Returns a pointer to the handle of the newly created child CD; returns 0 on an error.

Parameters

`in`	*color*	The "color" of the new child to which this task will belong
`in`	*num_tasks_in_↩ color*	The total number of tasks that are collectively creating the child numbered "color"; the collective waits for this number of tasks to arrive before creating the child
`in`	*name*	Optional user-specified name that can be used to "re-create" the same CD object if it was not destroyed yet; useful for resuing preserved state in CD trees that are not loop based.
`in`	*type*	Strict or relaxed
`in`	*error_name_↩ mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in`	*error_loc_mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in,out`	*error*	Pointer for error return value (kOK on success and kError on failure); no error value returned if error=0.

0.8.7.2.6 CDErrT cd::CDHandle::Destroy (bool *collective* = `false`)

Destroys a CD.

Destroys a CD by removing it from the CD tree and deleting all its data structures. Once Destroy() is called, **additional attempts to destroy the same CD instance may result in undefined behavior**. Destroy() need not be a collective operation because it typically comes after Complete(). However, a single task must call `Destroy(cd_object=true)` while the rest call `Destroy(cd_object=false)`.

Returns

May return kError if instance was already destroyed, but may also return kOK in such a case.

Parameters

`in`	*collective*	if `true`, destroy is done as a collective across all tasks that created the CD; otherwise the behavior is that only one task destroys the actual object while the rest just delete the local CDHandle.

0.8.7.2.7 CDNameT cd::CDHandle::GetName ( )

Get the name/location of this CD.

The CDName is a (level, number_within_level) tuple.

Returns

the name/location of the CD

0.8.7.2.8 CDHandle∗ cd::CDHandle::GetParent ( )

Get CDHandle to this CD's parent.

Returns

Pointer to CDHandle of parent

0.8.8 Detection and Recovery Methods

Classes

- class RecoverObject

  Recovery method that can be inherited and specialized by user.

Functions

- CDErrT cd::CDHandle::CDAssert (bool test_true, const SysErrT ∗error_to_report=0)

  User-provided detection function for failing a CD.
- CDErrT cd::CDHandle::CDAssertFail (bool test_true, const SysErrT ∗error_to_report=0)

  User-provided detection function for failing a CD.
- CDErrT cd::CDHandle::CDAssertNotify (bool test_true, const SysErrT ∗error_to_report=0)

  User-provided detection function for failing a CD.
- std::vector< SysErrT > cd::CDHandle::Detect (CDErrT ∗err_ret_val=0)

  Check whether any errors occurred while CD the executed.
- CDErrT cd::CDHandle::RegisterDetection (uint system_name_mask, uint system_loc_mask,)

  Declare that this CD can detect certain errors/failures by user-defined detectors.
- CDErrT cd::CDHandle::RegisterRecovery (uint error_name_mask, uint error_loc_mask, RecoverObject ∗recover_object=0)

  Register that this CD can recover from certain errors/failures.

0.8.8.1 Detailed Description

0.8.8.2 Function Documentation

0.8.8.2.1 CDErrT cd::CDHandle::CDAssert (bool *test_true*, const SysErrT ∗ *error_to_report* = 0)

User-provided detection function for failing a CD.

A user may call CDAssert() at any time during CD execution to assert correct execution behavior. If the test fails, the CD fails and must recover.

The CD runtime implementation may choose whether the CD fails at the point that the CDAssert() fails or whether the assertion failure is registered but only acted upon during the CD detection phase.

Returns

kOK when the assertion call completed successfully (regardless of whether the test was true or false) and kError if the action taken by the runtime on CDAssert() failure did not succeed.

Parameters

`in`	*test_true*	Boolean to be asserted as true.
`in,out`	*error_to_report*	An optional error report that will be used during recovery and for system diagnostics.

0.8.8.2.2 CDErrT cd::CDHandle::CDAssertFail (bool *test_true*, const SysErrT $*$ *error_to_report* = 0)

User-provided detection function for failing a CD.

A user may call CDAssertFail() at any time during CD execution to assert correct execution behavior. If the test fails, the CD fails and must recover.

CDAssertFail() fails immediately and calls recovery.

Returns

kOK when the assertion call completed successfully (regardless of whether the test was true or false) and kError if the action taken by the runtime on CDAssert() failure did not succeed.

Warning

May not be implemented yet.

Parameters

`in`	*test_true*	Boolean to be asserted as true.
`in,out`	*error_to_report*	An optional error report that will be used during recovery and for system diagnostics.

0.8.8.2.3 CDErrT cd::CDHandle::CDAssertNotify (bool *test_true*, const SysErrT $*$ *error_to_report* = 0)

User-provided detection function for failing a CD.

A user may call CDAssert() at any time during CD execution to assert correct execution behavior. If the test fails, the CD fails and must recover.

CDAssertNotify() registers the assertion failure, which is only acted upon during the CD detection phase.

Returns

kOK when the assertion call completed successfully (regardless of whether the test was true or false) and kError if the action taken by the runtime on CDAssert() failure did not succeed.

Parameters

`in`	*test_true*	Boolean to be asserted as true.
`in,out`	*error_to_report*	An optional error report that will be used during recovery and for system diagnostics.

0.8.8.2.4 std::vector$<$SysErrT$>$ cd::CDHandle::Detect (CDErrT $*$ *err_ret_val* = 0)

Check whether any errors occurred while CD the executed.

Only checks for those errors that the CD registered for, This function is only used for those errors that are logged during execution and not those that require immediate recovery.

This requires more thought and a more precise description.

Returns

any errors or failures detected during this CDs execution.

Parameters

`in,out`	*err_ret_val*	Pointer to a variable for optionally returning a CD runtime error code indicating some bug with Detect().

0.8.8.2.5 CDErrT cd::CDHandle::RegisterDetection (uint *system_name_mask*, uint *system_loc_mask*)

Declare that this CD can detect certain errors/failures by user-defined detectors.

The intent of this method is to specify to the autotuner that detection is possible. This is needed in order to balance between fine-grained and coarse-grained CDs and associated recovery.

Returns

kOK on success.

Parameters

`in`	*system_name_↩ mask*	each `1` in the mask indicates that this CD should be able to detect any errors that are meaningful to the application (in the error type mask).
`in`	*system_loc_↩ mask*	each `1` in the mask indicates that this CD should be able to detect any errors that are meaningful to the application (in the error type mask).

0.8.8.2.6 CDErrT cd::CDHandle::RegisterRecovery (uint *error_name_mask*, uint *error_loc_mask*, **RecoverObject** ∗ *recover_object* = `0`)

Register that this CD can recover from certain errors/failures.

This method serves two purposes:

It extends the specification of intent to recover provided in

Create().

It enables registering a customized recovery routine by

inheriting from the RecoverObject class.

Returns

kOK on success.

See also

Create(), RecoverObject

Todo Does registering recovery also imply turning on detection? Or is that done purely through RequireError↩ Probability()?

Parameters

`in`	*error_name_↩ mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in`	*error_loc_mask*	each `1` in the mask indicates that this CD should be able to recover from that error type.
`in`	*recover_object*	pointer to an object that contains the customized recovery routine; if unspecified, default recovery is used.

0.8.9 Methods for Interacting with the CD Framework and Tuner

Functions

- float cd::CDHandle::GetErrorProbability (SysErrT error_type, uint error_num,)

  Ask the CD framework to estimate error/fault rate.
- float cd::CDHandle::RequireErrorProbability (SysErrT error_type, uint error_num, float probability, bool fail_↩ over=true)

  Request the CD framework to reach a certain error/failure probability.

0.8.9.1 Detailed Description

Todo What about specifying leniant communication-related errors for relaxed-CDs context?

0.8.9.2 Function Documentation

0.8.9.2.1 float cd::CDHandle::GetErrorProbability (**SysErrT** *error_type,* uint *error_num*)

Ask the CD framework to estimate error/fault rate.

Each CD will experience a certain rate of failure/error for different failure/error mechanisms. These rates depend on the system, the duration of the CD, its memory footprint, etc. The CD framework can estimate the expected rate of each fault mechanism given its knowledge of the CD and system properties.

Returns

probability that the queried number of error/failure of the type queried will occur during the execution of this CD.

Note

Should this be some rate instead? Seems like it would be easier for the programmer to deal with number and probability, but is it?

Todo Decide on rate vs. number+probability

Parameters

`in`	*error_type*	Type of
	error_num	error/failure queried [in] Number of errors/failures queried.

0.8.9.2.2 float cd::CDHandle::RequireErrorProbability (**SysErrT** *error_type,* uint *error_num,* float *probability,* bool *fail_over* = `true`)

Request the CD framework to reach a certain error/failure probability.

Each CD will experience a certain rate of failure/error for different failure/error mechanisms. These rates depend on the system, the duration of the CD, its memory footprint, etc. The CD framework may be able to apply automatic resilience techniques, such as replication, to ensure certain errors/failures will be tolerated (or simply) detected.

Returns

probability that at least `error_num` errors/failurse of the type queried will occur during the execution of this CD, *after* the automatic techniques are applied. Should be less than or equal to requested `probability` if successful.

Note

Should this be some rate instead? Seems like it would be easier for the programmer to deal with number and probability, but is it?

Todo Decide on rate vs. number+probability

Parameters

`in`	*error_type*	Type of error/failure queried.
`in`	*error_num*	Number of errors/failures queried.
`in`	*probability*	Requested maximum probability of `num_errors` errors/failures not being detected or even occurring during CD execution.
`in`	*fail_over*	Should redundancy be added just to detect the specified error type (`false`) or should enough redundancy be added to tolerate the error (fail-over/forward-error-correction/...)

0.8.10 Profiler-related methods

Functions

- CDErrT cd::CDHandle::CDProfileStartPhase (bool collective=true, char ∗phase_name=0)

  Notify the CD Profiler that the application is entering a different execution phase.

0.8.10.1 Detailed Description

0.8.10.2 Function Documentation

0.8.10.2.1 CDErrT cd::CDHandle::CDProfileStartPhase (bool *collective* = `true`, char ∗ *phase_name* = `0`)

Notify the CD Profiler that the application is entering a different execution phase.

The CD Profiler attempts to collect information and build the CD tree corresponding to the application. Because the execution of each specific CD varies even when the code run and data processed is semantically the same, the profiler averages out the behavior of each CD in the tree as it gets repeatedly executed (execution time, preservation volumes, ...). The CDProfileStartPhase() notifies the profiler that a new execution phase is starting and that averaging should reset with the previous values maintained as a different very-coarse-grained sequential CD under the root CD. If the application repeats certain phases for which statistics should be aggregated, a state name should be provided.

Returns

Returns kOK if no error.

Parameters

collective	Collective across all tasks in this CD?
phase_name	Name of new phase

0.8.11 CD Event Functions for Non-Blocking Calls

Functions

- CDErrT cd::CDEvent::Wait (void)

  Blocking call waiting on the event to complete.
- bool cd::CDEvent::Test (void)

  Non-blocking call to test whether the event completed.

0.8.11.1 Detailed Description

0.8.11.2 Function Documentation

0.8.11.2.1 bool cd::CDEvent::Test (void)

Non-blocking call to test whether the event completed.

Returns

`true` if complete and `false` is any chained events not yet done.

0.8.11.2.2 CDErrT cd::CDEvent::Wait (void)

Blocking call waiting on the event to complete.

Once CDEvent::Wait() returns, the event is empty (uninitialized), as if it has been explicitly CDEvent::Reset().

Returns

kOK on success and kError if the event timed out

0.8.12 Internal Functions for Customizable Recovery

Functions

- virtual bool CDInternalPtr::InternalCanRecover (uint error_name_mask, unit error_location_mask)

  Method to test if this CD can recover from an error/location mask.
- virtual void CDInternalPtr::InternalReexecute ()

  Reexecute-style default recovery.
- virtual void CDInternalPtr::InternalEscalate (uint error_name_mask, unit error_location_mask, std::vector< Sys↩ ErrT > errorsuint error_name_mask,)

  Escalate error/failure to parent.

0.8.12.1 Detailed Description

0.8.12.2 Function Documentation

0.8.12.2.1 virtual bool CDInternalPtr::InternalCanRecover (uint *error_name_mask,* unit *error_location_mask*) `[virtual]`

Method to test if this CD can recover from an error/location mask.

Returns

Returns `true` is can recover and `false` otherwise.

See also

Create(), SysErrNameT, SysErrLocT

Parameters

`in`	*error_name_↩ mask*	Mask of all error/fail types that require recovery
`in`	*error_location_↩ mask*	Mask of all error/fail locations that require recovery

0.8.12.2.2 virtual void CDInternalPtr::InternalEscalate (uint *error_name_mask,* unit *error_location_mask,* std::vector< SysErrT > errorsuint *error_name_mask*) `[virtual]`

Escalate error/failure to parent.

Internal method used by Recover() to escalate errors/failures that cannot be handled.

Parameters

`in`	*error_name_↩ mask*	Mask of all error/fail types that require recovery
`in`	*error_location_↩ mask*	Mask of all error/fail locations that require recovery
`in`	*error_name_↩ mask*	Errors/failures to recover from (typically just one).

0.8.12.2.3 virtual void CDInternalPtr::InternalReexecute ( ) [virtual]

Reexecute-style default recovery.

Default recovery method of reexecuting the CD while restoring data as execution proceeds through Preserve() calls. **The alternative** is to have an explicit restore phase that restores all the data, but that places constraints on overlapping regions of preservation intermixed with modifications and requires more sophisticated metadata.

Todo Discuss other aspects of reexecution (e.g., logging). [FIXME] Discuss other aspects of reexecution (e.g., logging).

0.9 Namespace Documentation

0.9.1 cd Namespace Reference

Containment Domains namespace for global functions, types, and main interface.

Classes

- class CDEvent

    An object that provides an event identifier to a non-blocking CD runtime call.
- class CDHandle

    An object that provides a handle to a specific CD instance.
- struct CDNameT

    A type to uniquely name a CD in the tree.
- class DegradedMemErrInfo

    Interface to degraded memory error information.
- class RegenObject

    Interface for specifying regeneration functions for preserve/restore.
- class SoftMemErrInfo

    Interface to soft memory error information.
- struct SysErrT

    Type for specifying errors and failure.

Enumerations

- enum CDModeT { kStrict =0, kRelaxed }

    Type for specifying whether a CD is strict or relaxed.
- enum CDExecutionModeT { kExec =0, kReexec }

    Type for specifying whether the current CD is executing for the first time or is currently reexecuting as part of recovery.
- enum PreserveUseT { kUnsure =0, kReadOnly = 1, kReadWrite = 2 }

    Type to indicate whether preserved data is from read-only or potentially read/write application data.
- enum PGASUsageT { kShared = 0, kPotentiallyShared, kPrivatized, kPrivate }

    Different types of PGAS memory behavior for relaxed CDs.
- enum SysErrNameT {
  kOK =0, kSoftMem = 0b1, kDegradedMem = 0b01, kSoftComm = 0b001,
  kDegradedComm = 0b0001, kSoftComp = 0b00001, kDegradedResource =0b000001, kHardResource = 0b0000001,
  kFileSys = 0b00000001 }

    Type for specifying system errors and failure names.
- enum SysErrLocT {
  kOK =0, kIntraCore = 0b1, kCore = 0b01, kProc = 0b001,
  kNode = 0b0001, kModule = 0b00001, kCabinet = 0b000001, kCabinetGroup =0b0000001,
  kSystem = 0b00000001 }

    Type for specifying errors and failure location names.
- enum CDErrT { kOK =0, kAlreadyInit, kError }

    Type for specifying error return codes from an API call – signifies some failure of the API call itself, not a system failure.
- enum PreserveMechanismT { kCopy =0b001, kRef =0b010, kRegen =0b100 }

    Type for specifying preservation methods.

Functions

- uint DeclareErrName (const char ∗name_string)

    Create a new error/failure type name.
- CDErrT UndeclareErrName (uint error_name_id)

    Free a name that was created with DeclareErrorName()
- uint DeclareErrLoc (const char ∗name_string)

    Create a new error/failure type name.
- CDErrT UndeclareErrLoc (uint error_name_id) class SysErrInfo

    Free a name that was created with DeclareErrLoc()
- CDHandle ∗ Init (bool collective=trueCDErrT ∗error=0)

    Initialize the CD runtime.
- CDHandle ∗ GetRootCD ()

    Accessor function to root CD of the application.
- CDHandle ∗ GetCurrentCD ()

    Accessor function to current active CD.
- CDErrT SetCurrentCD (const CDHandle ∗cd)

    Accessor function for setting the current active CD.

0.9.1.1 Detailed Description

Containment Domains namespace for global functions, types, and main interface.

All user-visible CD API calls and definitions are under the CD namespace. Internal CD API implementation components are under a separate cd_internal namespace; the CDInternal class of namespace cd serves as an interface where necessary.

0.10 Class Documentation

0.10.1 cd::CDEvent Class Reference

An object that provides an event identifier to a non-blocking CD runtime call.

```
#include <cd.h>
```

Public Member Functions

- CDErrT Wait (void)

    Blocking call waiting on the event to complete.
- bool Test (void)

    Non-blocking call to test whether the event completed.

Protected Attributes

- cd_internal::CDEvent event_

0.10.1.1 Detailed Description

An object that provides an event identifier to a non-blocking CD runtime call.

This is basically just an internal event handle that the user can wait on when trying to make a non-blocking call.

Events are automatically chained (see CDHandle::Preserve()),

Note

We do not rely on C++11 async/futures here because the API is meant to be somewhat portable to other languages.

0.10.1.2 Member Data Documentation

0.10.1.2.1 cd_internal::CDEvent cd::CDEvent::event_ `[protected]`

The implementation-specific accessor to the actual CD event; not visible to user

The documentation for this class was generated from the following file:

- cd.h

0.10.2 cd::CDHandle Class Reference

An object that provides a handle to a specific CD instance.

`#include <cd.h>`

Public Member Functions

- CDHandle ∗ Create (char ∗name=0, CDModeT type=kStrict, uint error_name_mask=0, uint error_loc_mask=0, CDErrT ∗error=0)

  Single-task non-collective Create.
- CDHandle ∗ Create (uint_t color, uint_t num_tasks_in_color, char ∗name=0, CDModeT type=kStrict, uint error↩_name_mask=0, uint error_loc_mask=0, CDErrT ∗error=0)

  Collective Create.
- CDHandle ∗ CreateAndBegin (uint_t color, uint_t num_tasks_in_color, char ∗name=0, CDModeT type=kStrict, uint error_name_mask=0, uint error_loc_mask=0, CDErrT ∗error=0)

  Collective Create+Begin.
- CDErrT Destroy (bool collective=false)

  Destroys a CD.
- CDErrT Begin (bool collective=true)

  Begins a CD.
- CDErrT Complete (bool collective=true, bool update_preservations,)

  Completes a CD.
- CDNameT GetName ()

  Get the name/location of this CD.
- CDHandle ∗ GetParent ()

  Get CDHandle to this CD's parent.

- CDErrT Preserve (void ∗data_ptr, uint64_t len, uint_t preserve_mask=kCopy, const char ∗my_name=0, const char ∗ref_name=0, uint_64t ref_offset=0, const RegenObject ∗regen_object=0, PreserveUseT data_usage=k←Unsure)

  Preserve data to be restored when recovering (typically reexecuting the CD from right after its Begin() call.
- CDErrT Preserve (CDEvent &cd_event, void ∗data_ptr, uint64_t len, uint_t preserve_mask=kCopy, const char ∗my_name=0, const char ∗ref_name=0, uint_64t ref_offset=0, const RegenObject ∗regen_object=0, Preserve←UseT data_usage=kUnsure)

  Non-blocking preserve data to be restored when recovering (typically reexecuting the CD from right after its Begin() call.
- CDErrT CDAssert (bool test_true, const SysErrT ∗error_to_report=0)

  User-provided detection function for failing a CD.
- CDErrT CDAssertFail (bool test_true, const SysErrT ∗error_to_report=0)

  User-provided detection function for failing a CD.
- CDErrT CDAssertNotify (bool test_true, const SysErrT ∗error_to_report=0)

  User-provided detection function for failing a CD.
- std::vector< SysErrT > Detect (CDErrT ∗err_ret_val=0)

  Check whether any errors occurred while CD the executed.
- CDErrT RegisterDetection (uint system_name_mask, uint system_loc_mask,)

  Declare that this CD can detect certain errors/failures by user-defined detectors.
- CDErrT RegisterRecovery (uint error_name_mask, uint error_loc_mask, RecoverObject ∗recover_object=0)

  Register that this CD can recover from certain errors/failures.
- float GetErrorProbability (SysErrT error_type, uint error_num,)

  Ask the CD framework to estimate error/fault rate.
- float RequireErrorProbability (SysErrT error_type, uint error_num, float probability, bool fail_over=true)

  Request the CD framework to reach a certain error/failure probability.
- CDErrT SetPGASUsage (void ∗data_ptr, uint64_t len, PGASUsageT region_type=kShared)

  Declare how a region of memory behaves within this CD (for Relaxed CDs)
- CDErrT SetPGASOwnerWrites (void ∗data_ptruint64_t len, bool owner_writes=true)

  Simplify optimization of discarding relaxed CD log entries.
- CDErrT CDProfileStartPhase (bool collective=true, char ∗phase_name=0)

  Notify the CD Profiler that the application is entering a different execution phase.

Protected Attributes

- CDInternalPtr cd_instance_
- bool destroy_cd_object_hint_

0.10.2.1 Detailed Description

An object that provides a handle to a specific CD instance.

All usage of CDs (other than CD Init Functions and cd_accessor_funcs) is done by utilizing a handle to a particular CD instance within the CD tree. The CDHandle provides an implementation- and location-independent accessor for CD operation.

Most calls currently only have blocking versions. Which should also be non-blocking other than Preserve?

0.10.2.2 Member Data Documentation

0.10.2.2.1 CDInternalPtr cd::CDHandle::cd_instance_ [protected]

The implementation-specific accessor to the actual CD instance; not visible to user

0.10.2.2.2 bool cd::CDHandle::destroy_cd_object_hint_ `[protected]`

Hint set by a collective Create() so that destroying the object is called by only a single task.

The documentation for this class was generated from the following file:

- cd.h

0.10.3 CDInternalPtr Class Reference

A class that represents the interface to the internal implementation of an actual CD.

`#include <cd.h>`

Public Member Functions

- virtual bool InternalCanRecover (uint error_name_mask, unit error_location_mask)

  Method to test if this CD can recover from an error/location mask.
- virtual void InternalReexecute ()

  Reexecute-style default recovery.
- virtual void InternalEscalate (uint error_name_mask, unit error_location_mask, std::vector< SysErrT > errorsuint error_name_mask,)

  Escalate error/failure to parent.

0.10.3.1 Detailed Description

A class that represents the interface to the internal implementation of an actual CD.

The documentation for this class was generated from the following file:

- cd.h

0.10.4 cd::CDNameT Struct Reference

A type to uniquely name a CD in the tree.

`#include <cd.h>`

Public Attributes

- uint level

  Level within the tree (root=0)
- uint number

  Unique ID within level.

0.10.4.1 Detailed Description

A type to uniquely name a CD in the tree.

A CD name consists of its level in the CD tree (root=0)and the its ID, or sequence number, within that level.

Note

Alternatives:

- The alternative of simply a unique name misses the idea of levels in the tree; the idea of hierarchy is central to CDs so this is a bad alternative.
- The alternative of naming a CD by its entire "branch" leads to requiring the name to be parsed to identify the level; the level is typically the most crucial information so this seems unduly complex.
- A third alternative is to store the branch information as a std::vector, so the vector's length is the level. However, this means the name is rather long.

Todo Decide on whether to represent a CD name as (level, num) or as `typedef std::vector<uint> CD↩ NameT;` to represent the path through the tree branches.

The documentation for this struct was generated from the following file:

- cd.h

0.10.5 cd::DegradedMemErrInfo Class Reference

Interface to degraded memory error information.

`#include <cd.h>`

Public Member Functions

- std::vector< uint64_t > get_pa_starts ()

  Starting physical addresses.
- std::vector< uint64_t > get_va_starts ()

  Starting virtual addresses.
- std::vector< uint64_t > get_lengths ()

  Lengths of affected regions.

Protected Attributes

- std::vector< uint64_t > pa_starts_

  Starting physical addresses.
- std::vector< uint64_t > va_starts_

  Starting virtual addresses.
- std::vector< uint64_t > lengths_

  Lengths of affected regions.

0.10.5.1 Detailed Description

Interface to degraded memory error information.

This is meant to potentially be extended.

The documentation for this class was generated from the following file:

- cd.h

0.10.6 RecoverObject Class Reference

Recovery method that can be inherited and specialized by user.

`#include <cd.h>`

Public Member Functions

- virtual void Recover (CDInternalPtr ∗cd_instance, uint error_name_mask, unit error_location_mask, std::vector< SysErrT > errors)

  Recover method to be specialized by inheriting and overloading.

0.10.6.1 Detailed Description

Recovery method that can be inherited and specialized by user.

The purpose of RecoverObject is to provide an interface to enable a programer to create custom Recover routines. The idea is that for each CD, each error type+location may be bound to a specialized recovery routine, which is expressed through a Recover object. The Recover object inherits the default RecoverObject and extends or replaces the default restore+reexecute recovery method.

Todo Write some example for custom recovery (see GVR interpolation example, although they do it between versions).

See also

CDHandle::RegisterRecovery()

0.10.6.2 Member Function Documentation

0.10.6.2.1 virtual void RecoverObject::Recover (CDInternalPtr ∗ *cd_instance*, uint *error_name_mask*, unit *error_location_mask*, std::vector< SysErrT > *errors*) `[inline]`, `[virtual]`

Recover method to be specialized by inheriting and overloading.

Recover uses methods that are internal to the CD itself and should only be called by customized recovery routines that inherit from RecoverObject and modify the Recover() method.

Parameters

	cd_instance	A pointer to the actual CD instance so that the internal methods can be called.
`in`	*error_name_mask*	Mask of all error/fail types that require recovery
`in`	*error_location_mask*	Mask of all error/fail locations that require recovery
`in`	*errors*	Errors/failures to recover from (typically just one).

The documentation for this class was generated from the following file:

- cd.h

0.10.7 cd::RegenObject Class Reference

Interface for specifying regeneration functions for preserve/restore.

`#include <cd.h>`

Public Member Functions

- virtual CDErrT Regenerate (void ∗data_ptr, uint64_t len)=0

  Pure virtual interface function for regenerating data as restoration type.

0.10.7.1 Detailed Description

Interface for specifying regeneration functions for preserve/restore.

An interface for a data regeneration function that can be used to restore "preserved" data instead of making a copy of the data to be preserved.

See also

PreserveMethodT, CDHandle::Preserve()

0.10.7.2 Member Function Documentation

0.10.7.2.1 virtual **CDErrT** cd::RegenObject::Regenerate (void ∗ *data_ptr,* uint64_t *len*) `[pure virtual]`

Pure virtual interface function for regenerating data as restoration type.

Must be implemented by programmer.

Returns

Should return a CD error value if regeneration is not successful.

Parameters

data_ptr	Pointer to data that is to be regenerated.
len	Length of data to be regenerated.

The documentation for this class was generated from the following file:

- cd.h

0.10.8 cd::SoftMemErrInfo Class Reference

Interface to soft memory error information.

`#include <cd.h>`

Public Member Functions

- uint64_t get_pa_start ()

  Starting physical address.
- uint64_t get_va_start ()

  Starting virtual address.
- uint64_t get_length ()

  Length of affected access.
- char[] get_data ()

  Data value read (erroneous)

- uint64_t get_syndrome_len ()

  Length of syndrome.
- char[ ] get_syndrome ()

  Value of syndrome.

Protected Attributes

- uint64_t pa_start_

  Starting physical address.
- uint64_t va_start_

  Starting virtual address.
- uint64_t length_

  Length of affected access.
- char ∗ data_

  Data value read (erroneous)
- uint64_t syndrome_len_

  Length of syndrome.
- char ∗ syndrome_

  Value of syndrome.

0.10.8.1 Detailed Description

Interface to soft memory error information.

This is meant to potentially be extended.

The documentation for this class was generated from the following file:

- cd.h

0.10.9 cd::SysErrT Struct Reference

Type for specifying errors and failure.

`#include <cd.h>`

Public Attributes

- SysErrNameT error_name_

  Name of error.
- SysErrLocT error_location_

  Location of error.
- SysErrInfo error_info_

  Error-specific extra information.

0.10.9.1 Detailed Description

Type for specifying errors and failure.

This type represents the interface between the user and the system with respect to errors and failures. We considered doing an extensible class hierarchy like GVR, but ended up with predefined bitvector constants because of the pain involved in setting up and using deep class hierarchies. However, the bitmask way is dangerously narrow and may lead to less portable (and less future-proof code). Basically we chose C over C++ style here :-(

This needs more thought

The documentation for this struct was generated from the following file:

- cd.h

0.11 File Documentation

0.11.1 cd.h File Reference

Containment Domains API v0.1 (C++)

```
#include <vector>
```

Classes

- struct cd::CDNameT

  A type to uniquely name a CD in the tree.
- class cd::SoftMemErrInfo

  Interface to soft memory error information.
- class cd::DegradedMemErrInfo

  Interface to degraded memory error information.
- struct cd::SysErrT

  Type for specifying errors and failure.
- class cd::RegenObject

  Interface for specifying regeneration functions for preserve/restore.
- class cd::CDHandle

  An object that provides a handle to a specific CD instance.
- class cd::CDEvent

  An object that provides an event identifier to a non-blocking CD runtime call.
- class CDInternalPtr

  A class that represents the interface to the internal implementation of an actual CD.
- class RecoverObject

  Recovery method that can be inherited and specialized by user.

Namespaces

- cd

  Containment Domains namespace for global functions, types, and main interface.

Enumerations

- enum cd::CDModeT { cd::kStrict =0, cd::kRelaxed }

  Type for specifying whether a CD is strict or relaxed.
- enum cd::CDExecutionModeT { cd::kExec =0, cd::kReexec }

  Type for specifying whether the current CD is executing for the first time or is currently reexecuting as part of recovery.
- enum cd::PreserveUseT { cd::kUnsure =0, cd::kReadOnly = 1, cd::kReadWrite = 2 }

  Type to indicate whether preserved data is from read-only or potentially read/write application data.
- enum cd::PGASUsageT { cd::kShared = 0, cd::kPotentiallyShared, cd::kPrivatized, cd::kPrivate }

  Different types of PGAS memory behavior for relaxed CDs.
- enum cd::SysErrNameT {
  cd::kOK =0, cd::kSoftMem = 0b1, cd::kDegradedMem = 0b01, cd::kSoftComm = 0b001,
  cd::kDegradedComm = 0b0001, cd::kSoftComp = 0b00001, cd::kDegradedResource =0b000001, cd::kHard←
  Resource = 0b0000001,
  cd::kFileSys = 0b00000001 }

  Type for specifying system errors and failure names.
- enum cd::SysErrLocT {
  cd::kOK =0, cd::kIntraCore = 0b1, cd::kCore = 0b01, cd::kProc = 0b001,
  cd::kNode = 0b0001, cd::kModule = 0b00001, cd::kCabinet = 0b000001, cd::kCabinetGroup =0b0000001,
  cd::kSystem = 0b00000001 }

  Type for specifying errors and failure location names.
- enum cd::CDErrT { cd::kOK =0, cd::kAlreadyInit, cd::kError }

  Type for specifying error return codes from an API call – signifies some failure of the API call itself, not a system failure.
- enum cd::PreserveMechanismT { cd::kCopy =0b001, cd::kRef =0b010, cd::kRegen =0b100 }

  Type for specifying preservation methods.

Functions

- uint cd::DeclareErrName (const char ∗name_string)

  Create a new error/failure type name.
- CDErrT cd::UndeclareErrName (uint error_name_id)

  Free a name that was created with DeclareErrorName()
- uint cd::DeclareErrLoc (const char ∗name_string)

  Create a new error/failure type name.
- CDErrT cd::UndeclareErrLoc (uint error_name_id) class SysErrInfo

  Free a name that was created with DeclareErrLoc()
- CDHandle ∗ cd::Init (bool collective=trueCDErrT ∗error=0)

  Initialize the CD runtime.
- CDHandle ∗ cd::GetRootCD ()

  Accessor function to root CD of the application.
- CDHandle ∗ cd::GetCurrentCD ()

  Accessor function to current active CD.
- CDErrT cd::SetCurrentCD (const CDHandle ∗cd)

  Accessor function for setting the current active CD.

0.11.1.1 Detailed Description

Containment Domains API v0.1 (C++)

Author

Kyushick Lee, Jinsuk Chung, Song Zhang, Seong-Lyong Gong, Derong Liu, Mattan Erez

Date

March 2014

0.12 Example Documentation

0.12.1 spmv.cc

```
/*
Copyright 2014, The University of Texas at Austin
All rights reserved.

THIS FILE IS PART OF THE CONTAINMENT DOMAINS RUNTIME LIBRARY

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
  COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE.
*/

/*
 * @file spmv.cc
 * @author Mattan Erez
 * @date March 2014
 *
 * @brief Hierarchical SpMV CD Example
 *
 * The SpMV computation consists of iteratively multiplying a constant
 * matrix by an input vector.  The resultant vector is then used as the
 * input for the next iteration.  We assume that the matrix and vector
 * are block partitioned and assigned to multiple nodes and cores.  This
 * simple application demonstrates many of the features of CDs and how
 * they can be used to express efficient resilience.
 *
 * One of the advantages of containment domains is that preservation and
 * recovery can be tailored to exploit natural redundancy within the
 * machine.  A CD does not need to fully preserve its inputs at the
 * domain boundary; partial preservation may be utilized to increase
 * efficiency if an input naturally resides in multiple locations.
 * Examples for optimizing preserve/restore/recover routines include
 * restoring data from sibling CDs or other nodes which already have a
 * copy of the data for algorithmic reasons.
 *
 * Hierarchical SpMV exhibits natural redundancy which can be
 * exploited through partial preservation and specialized
 * recovery. The input vector is distributed in such a way that
 * redundant copies of the vector are naturally distributed throughout
 * the machine. This is because there are \f$ N_0 \times N_0 \f$ fine-grained
 * sub-blocks of the matrix, but only \f$ N_0 \f$ sub-blocks in the vector.
 *
 * This is a hierarchical/recursive form of SpMV that uses some
 * pseudocode just to demonstrate the usage of the CD API
 */

#include "cd.h"

class VInRegen : public RegenType {
public:
  VInRegen(uint64_t task_num, uint subpartition) {
    task_num_ = task_num; subparition_ = subpartition;
  }

  CDErrType Regenerate(void* data_ptr, uint64_t len) {
    // During recovery (re-execution) Preserve acts like Restore
    // Writing this regeneration, which is really recovering data from
    // a sibling that has the same copy of the input vector is
    // difficult without assuming a specific parallelism
    // runtime. Unfortunately, both my MPI and UPC are rusty so I
    // can't do it right now. The idea is that we know which sibling
    // has data that we need based on the matrix partitioning and that
    // we know our subpartition number. We can then use the
    // parallelism runtime (or whoever tracked the task recursion) to
    // know which thread/rank/... this sibling is in, but we still
    // need to know its pointer to do a one-sided transfer of the data
    // because recovering this CD is independent of the sibling).
  }

protected:
  uint64_t task_num_;
  uint subpartition_;
};

/* @brief The recursive part that decomposes the problem for parallelism and containment.
 *
 * For simplicity, we assume that the input matrix has been
 * pre-partitioned to the appropriate number of levels to allow the
 * recursion to work correctly.
 *
 */
void SpMVRecurse(const SparseMatrix* matrix,
     const HierVector* v_in,
     HierVector*       v_out,
     const CDHandle* current_cd,
     uint  num_tasks
         ) {


  if (num_tasks > RECURSE_DEGREE) {
    uint tasks_per_child = num_tasks/RECURSE_DEGREE; // assume whole multiple
    for (int child=0; child < RECURSE_DEGREE; child++) {
      // assume that all iterations are all in parallel
      CDHandle* child_cd;
      // Creating the children CDs here so that we can more easily use
      // a collective mpi_comm_split-like Create method. This would be
      // easier if done internally by parallelism runtime/language
      child_cd = current_cd->CreateAndBegin(child, tasks_per_child);
      // Do some preservation
      CDEvent preserve_event;
      child_cd->Preserve(preserve_event,
       matrix->Subpartition(child), // Pointer to
       // start of subpartition within recursive matrix
        matrix->SubpartitionLen(child), // Length in
                // bytes of subpartition
        kCopy|kParent, // Can either create another
           // copy or use the parent's
           // preserved matrix with
           // appropriate offset
        "Matrix",
       "Matrix", matrix->PartitionOffset(),
       0,
       kReadOnly
       );
      // Regen object for input vector, assuming there is a
      // parallelism runtime that tracks recursion tree through task numbers
      VInRegen v_in_regen(ParRuntime::MyTaskNum(), child);
      child_cd->Preserve(preserve_event, // Chain this event
       v_in->Subpartition(child),
       v_in->SubpartitionLen(child),
       kCopy|kParent|kRegen,
       "vIn",
       "vIn", v_in->PartitionOffset(),
       &v_in_regen
       );
      // Do actual compute
      SpMVRecurse(matrix->Subpartition(child),
      v_in->Subpartition(child),
      v_out->Subpartition(child),
      child_cd, tasks_per_child);
      // Complete the CD
      preserve_event->Wait(); // Make sure preservation completed
      child_cd->Complete();
      child_cd->Destroy();
    }
  }
  else {
    for (int child=0; child < num_tasks; child++) {
      // assume that all iterations are all in parallel
      CDHandle* child_cd;
      CDHandle* child_cd = current_cd->Create();
      child_cd->Begin();
      // Do some preservation
      CDEvent preserve_event;
      child_cd->Preserve(preserve_event,
       matrix->Partition(), // Pointer to
       // start of subpartition within recursive matrix
       matrix->PartitionLen(), // Length in
       // bytes of subpartition
       kCopy|kParent, // Can either create another
       // copy or use the parent's
       // preserved matrix with
       // appropriate offset
       "Matrix",
       "Matrix", matrix->PartitionOffset(),
       0,
       kReadOnly
       );
      VInRegen v_in_regen(ParRuntime::MyTaskNum(), child);
      child_cd->Preserve(preserve_event, // Chain this event
       v_in->Partition(),
       v_in->SubpartitionLen(),
       kCopy|kParent|kRegen,
       "vIn",
       "vIn", v_in->PartitionOffset(),
       &v_in_regen
       );
      // Do actual compute
      SpMVLeaf(matrix->Subpartition(child),
         v_in->Subpartition(child),
         v_out->Subpartition(child),
         child_cd, tasks_per_child);
      child_cd->Complete();
      child_cd->Destroy();
    }
  }

  v_out->ReduceSubpartitions(num_tasks);
}

void SpMVLeaf(const SparseMatrix* matrix,
        const HierVector* v_in,
        HierVector*       v_out,
        const CDHandle* current_cd,
        uint  num_tasks
        ) {
  for (uint row=0; row < matrix->NumRows(); row++) {
    v_out[row] = 0.0;
    for (unit col = matrix->RowStart[row];
   col <  matrix->RowStart[row+1];
   col++) {
      uint prev_idx = 0;
      uint idx = matrix->Index[col];
      v_out[row] += matrix->NonZero[col]*v_in[idx];
      CDAssert(idx >= prev_idx); // data structure sanity check
      prev_idx = idx;
    }
  }
}
```

Index

CD-Related Definitions
  kExec, 7
  kReadOnly, 8
  kReadWrite, 8
  kReexec, 7
  kRelaxed, 7
  kStrict, 7
  kUnsure, 8
cd, 35

Detect
  Detection and Recovery Methods, 26
Detection and Recovery Methods, 25
  Detect, 26

Error Reporting, 11
  kAlreadyInit, 12
  kCabinet, 12
  kCabinetGroup, 12
  kCore, 12
  kDegradedComm, 13
  kDegradedMem, 13
  kDegradedResource, 13
  kError, 12
  kFileSys, 13
  kHardResource, 13
  kIntraCore, 12
  kModule, 12
  kNode, 12
  kOK, 12, 13
  kProc, 12
  kSoftComm, 13
  kSoftComp, 13
  kSoftMem, 13
  kSystem, 12

Internal Functions for Customizable Recovery, 33

kAlreadyInit
  Error Reporting, 12
kCabinet
  Error Reporting, 12
kCabinetGroup
  Error Reporting, 12
kCopy
  Preservation/Restoration Types and Methods, 15
kCore
  Error Reporting, 12
kDegradedComm
  Error Reporting, 13
kDegradedMem
  Error Reporting, 13
kDegradedResource
  Error Reporting, 13
kError
  Error Reporting, 12
kExec
  CD-Related Definitions, 7
kFileSys
  Error Reporting, 13
kHardResource
  Error Reporting, 13
kIntraCore
  Error Reporting, 12
kModule
  Error Reporting, 12
kNode
  Error Reporting, 12
kOK
  Error Reporting, 12, 13
kPotentiallyShared
  PGAS-Specific Methods and Types, 9
kPrivate
  PGAS-Specific Methods and Types, 9
kPrivatized
  PGAS-Specific Methods and Types, 9
kProc
  Error Reporting, 12
kReadOnly
  CD-Related Definitions, 8
kReadWrite
  CD-Related Definitions, 8
kReexec
  CD-Related Definitions, 7
kRef
  Preservation/Restoration Types and Methods, 15
kRegen
  Preservation/Restoration Types and Methods, 15
kRelaxed
  CD-Related Definitions, 7
kShared

    PGAS-Specific Methods and Types, 9
kSoftComm
    Error Reporting, 13
kSoftComp
    Error Reporting, 13
kSoftMem
    Error Reporting, 13
kStrict
    CD-Related Definitions, 7
kSystem
    Error Reporting, 12
kUnsure
    CD-Related Definitions, 8

PGAS-Specific Methods and Types
    kPotentiallyShared, 9
    kPrivate, 9
    kPrivatized, 9
    kShared, 9
Preservation/Restoration Types and Methods
    kCopy, 15
    kRef, 15
    kRegen, 15
Profiler-related methods, 31