SuperTrans

0.1f

Generated by Doxygen 1.5.5

Mon Sep 7 12:51:52 2009

Contents

Chapter 1

Main Page

Figure 1.1: SuperTrans

1.1 Introduction

SuperTrans is a cycle-accurate, detailed hardware transactional memory model that is capable of simulating all of the major dimensions of conflict detection and version management (eager-eager, eager-lazy, and lazy-lazy). SuperTrans is built on top of SESC (included), a cycle-accurate MIPS CMP simulator. SuperTrans has been used in previous research for workload characterization of hardware transactional memory systems and is particularly suited to this task as it allows for abstraction of the underlying model implementation. While the SuperTrans implementation of eager-eager and lazy-lazy systems is based on LogTM and TCC respectively, the model allows for abstraction of the specific overheads associated with tasks such as bus arbitration, NACK and backoff stall policies, conflict detection granularity, etc. This allows the researcher to gain insight into fundamental characteristics within and across design dimensions without being limited to any single design implementation

1.2 Installation

See documentation accompanied with the installation.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 cacheState Struct Reference

Cache Line State Container (R/W).

`#include <transCoherence.h>`

4.1.1 Detailed Description

Cache Line State Container (R/W).

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

- simulators/sesc/SESC-Gold/src/libtrans/**transCoherence.h**

4.2 GCMFinalRet Struct Reference

Structure used to store result of coherence request.

`#include <transCoherence.h>`

Data Fields

- long long **tuid**

  This allows tagging of DINST instructions with information as to whether the transaction is new, replayed, or subsumed.

4.2.1 Detailed Description

Structure used to store result of coherence request.

This is needed because we need additional information about the write set size for aborts/commits

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

- simulators/sesc/SESC-Gold/src/libtrans/**transCoherence.h**

4.3 memRef Struct Reference

memory reference structure used for transMemRef

`#include <transReport.h>`

4.3.1 Detailed Description

memory reference structure used for transMemRef

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

- simulators/sesc/SESC-Gold/src/libtrans/**transReport.h**

4.4 tmState Struct Reference

TM State Container.

`#include <transCoherence.h>`

4.4.1 Detailed Description

TM State Container.

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

- simulators/sesc/SESC-Gold/src/libtrans/**transCoherence.h**

4.5 transactionCache Class Reference

Transaction Cache.

`#include <transCache.h>`

Public Member Functions

- RAddr **findWordAddress** (RAddr addr)

  Find word boundary.

- IntRegValue **loadByte** (RAddr addr)

  Byte-size loads.

- double **loadDFP** (RAddr addr)

  Double prec. floating-point loads.

- float **loadFPWord** (RAddr addr)

  Floating-point loads.

- IntRegValue **loadHalfword** (RAddr addr)

  Half word-size loads.

- IntRegValue **loadUnsignedHalfword** (RAddr addr)

  Unsigned half word-size loads.

- IntRegValue **loadWord** (RAddr addr)

  Word-size loads.

- void **storeByte** (RAddr addr, IntRegValue value)

  Byte-size stores.

- void **storeDFP** (RAddr addr, unsigned long long value)

  Double prec. floating-point stores.

- void **storeFPWord** (RAddr addr, IntRegValue value)

  Floating-point stores.

- void **storeHalfWord** (RAddr addr, IntRegValue value)

  Half Word-size stores.

- void **storeWord** (RAddr addr, IntRegValue value)

  Word-size stores.

- **transactionCache** ()

  Default constructor.

- **~transactionCache** ()

  Default destructor.

Private Attributes

- map< RAddr, IntRegValue > **memMap**

  The Memory Map.

4.5.1 Detailed Description

Transaction Cache.

This class contains the methods and fields required for storing memory information about references inside of a transaction.

4.5.2 Member Function Documentation

4.5.2.1 RAddr transactionCache::findWordAddress (RAddr *addr*)

Find word boundary.

Parameters:

addr Real address

Returns:

Real address

4.5.2.2 IntRegValue transactionCache::loadByte (RAddr *addr*)

Byte-size loads.

Parameters:

addr Real address

Returns:

Memory value (int)

4.5.2.3 double transactionCache::loadDFP (RAddr *addr*)

Double prec. floating-point loads.

Parameters:

addr Real address

Returns:

Memory value (double prec.)

4.5.2.4 float transactionCache::loadFPWord (RAddr *addr*)

Floating-point loads.

Parameters:

addr Real address

Returns:

Memory contents (float)

4.5.2.5 IntRegValue transactionCache::loadHalfword (RAddr *addr*)

Half word-size loads.

Parameters:

addr Real address

Returns:

Memory value (int)

4.5.2.6 IntRegValue transactionCache::loadUnsignedHalfword (RAddr *addr*)

Unsigned half word-size loads.

Parameters:

addr Real address

Returns:

Memory value (int)

4.5.2.7 IntRegValue transactionCache::loadWord (RAddr *addr*)

Word-size loads.

Parameters:

addr Real address

Returns:

Memory value (int)

Full description.

4.5.2.8 void transactionCache::storeByte (RAddr *addr*, IntRegValue *value*)

Byte-size stores.

Parameters:

addr Real address

value Memory value (int)

4.5.2.9 void transactionCache::storeDFP (RAddr *addr*, unsigned long long *value*)

Double prec. floating-point stores.

Parameters:

addr Real address

value Memory value (64b)

4.5.2.10 void transactionCache::storeFPWord (RAddr *addr*, IntRegValue *value*)

Floating-point stores.

Parameters:

addr Real address

value Memory value (int)

4.5.2.11 void transactionCache::storeHalfWord (RAddr *addr*, IntRegValue *value*)

Half Word-size stores.

Parameters:

addr Real address

value Memory value (int)

4.5.2.12 void transactionCache::storeWord (RAddr *addr*, IntRegValue *value*)

Word-size stores.

Parameters:

addr Real address

value Memory value (int)

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

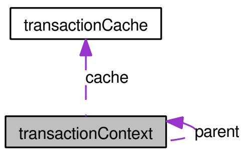
- simulators/sesc/SESC-Gold/src/libtrans/**transCache.h**
- simulators/sesc/SESC-Gold/src/libtrans/**transCache.cpp**

4.6 transactionContext Class Reference

transactional context

`#include <transContext.h>`

Collaboration diagram for transactionContext:



Public Member Functions

- void **abortTransaction** (thread_ptr pthread)

  Aborts a transaction.

- void **beginTransaction** (thread_ptr pthread, icode_ptr picode)

  Starts a transaction.

- void **cacheLB** (thread_ptr pthread, icode_ptr picode, RAddr raddr)

  load byte from TM cache

- void **cacheLDFP** (thread_ptr pthread, icode_ptr picode, RAddr raddr)

  load double prec floating point from TM cache

- void **cacheLHW** (thread_ptr pthread, icode_ptr picode, RAddr raddr)

  load half word from TM cache

- void **cacheLUB** (thread_ptr pthread, icode_ptr picode, RAddr raddr)

  load unsigned byte from TM cache

- void **cacheLUH** (thread_ptr pthread, icode_ptr picode, RAddr raddr)

  load unsigned half word from TM cache

- void **cacheLW** (thread_ptr pthread, icode_ptr picode, RAddr raddr)

  load word from TM cache

- void **cacheLWFP** (thread_ptr pthread, icode_ptr picode, RAddr raddr)

  load single prec floating point from TM cache

- void **cacheReadBuffer** (thread_ptr pthread, icode_ptr picode, char ∗buff, RAddr buffBegin, int count)

  read entire buffer from TM cache

- void **cacheSB** (thread_ptr pthread, icode_ptr picode, RAddr raddr, IntRegValue value)

  store byte to TM cache

- void **cacheSDFP** (thread_ptr pthread, icode_ptr picode, RAddr raddr, unsigned long long value)

  store double prec floating point to TM cache
- void **cacheSHW** (thread_ptr pthread, icode_ptr picode, RAddr raddr, IntRegValue value)

  store half word to TM cache
- void **cacheSW** (thread_ptr pthread, icode_ptr picode, RAddr raddr, IntRegValue value)

  store word to TM cache
- void **cacheSWFP** (thread_ptr pthread, icode_ptr picode, RAddr raddr, IntRegValue value)

  store single prec floating point to TM cache
- void **cacheWriteBuffer** (thread_ptr pthread, icode_ptr picode, char *buff, RAddr buff-Begin, int count)

  store entire buffer to TM cache
- bool **checkAbort** ()

  Checks to see if the transaction is aborted.
- void **commitTransaction** (thread_ptr pthread, icode_ptr picode)

  Commits a transaction.
- **transactionContext** (thread_ptr pthread, icode_ptr picode)

  Constructor.
- **transactionContext** ()

  Default constructor.
- ∼**transactionContext** ()

  Default destructor.

Private Member Functions

- void **createStall** (thread_ptr pthread, int stallLength)

  Stalls a thread for a given period.
- void **stallInstruction** (thread_ptr pthread, icode_ptr picode, int stallLength)

  Stalls the GCM for a given period.

4.6.1 Detailed Description

transactional context

Middleware between thread context and transactional cache/coherence protocol. Exactly one per dynamic transaction.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 transactionContext::transactionContext (thread_ptr *pthread*, icode_ptr *picode*)

Constructor.

Parameters:

pthread SESC thread pointer

picode Instruction code

4.6.3 Member Function Documentation

4.6.3.1 void transactionContext::abortTransaction (thread_ptr *pthread*)

Aborts a transaction.

Parameters:

pthread SESC thread pointer

4.6.3.2 void transactionContext::beginTransaction (thread_ptr *pthread*, icode_ptr *picode*)

Starts a transaction.

Parameters:

pthread SESC thread pointer

picode Instruction code

Set the BCFlag from retVal to indicate whether a Replay trans or not

Clear the aborting flag just in case it hasn't previously been cleared

Move instruction pointer to next instruction

Set the BCFlag to the retVal version (in this case it should indicate subsumed)

4.6.3.3 void transactionContext::cacheLB (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*)

load byte from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

4.6.3.4 void transactionContext::cacheLDFP (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*)

load double prec floating point from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

4.6.3.5 void transactionContext::cacheLHW (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*)

load half word from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

4.6.3.6 void transactionContext::cacheLUB (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*)

load unsigned byte from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

4.6.3.7 void transactionContext::cacheLUH (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*)

load unsigned half word from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

4.6.3.8 void transactionContext::cacheLW (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*)

load word from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

4.6.3.9 void transactionContext::cacheLWFP (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*)

load single prec floating point from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

4.6.3.10 void transactionContext::cacheReadBuffer (thread_ptr *pthread*, icode_ptr *picode*, char * *buff*, RAddr *buffBegin*, int *count*)

read entire buffer from TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

buff

buffBegin

count

4.6.3.11 void transactionContext::cacheSB (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*, IntRegValue *value*)

store byte to TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

value Cache line value

4.6.3.12 void transactionContext::cacheSDFP (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*, unsigned long long *value*)

store double prec floating point to TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

value Cache line value

4.6.3.13 void transactionContext::cacheSHW (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*, IntRegValue *value*)

store half word to TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

value Cache line value

4.6.3.14 void transactionContext::cacheSW (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*, IntRegValue *value*)

store word to TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

value Cache line value

4.6.3.15 void transactionContext::cacheSWFP (thread_ptr *pthread*, icode_ptr *picode*, RAddr *raddr*, IntRegValue *value*)

store single prec floating point to TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

raddr Real address

value Cache line value

4.6.3.16 void transactionContext::cacheWriteBuffer (thread_ptr *pthread*, icode_ptr *picode*, char * *buff*, RAddr *buffBegin*, int *count*)

store entire buffer to TM cache

Parameters:

pthread SESC thread pointer

picode Instruction code

buff

buffBegin

count

4.6.3.17 bool transactionContext::checkAbort ()

Checks to see if the transaction is aborted.

Returns:

Is this transaction aborted?

4.6.3.18 void transactionContext::commitTransaction (thread_ptr *pthread*, icode_ptr *picode*)

Commits a transaction.

Parameters:

pthread SESC thread pointer SESC thread pointer

picode Instruction code

We first delay during the commit

In the case of a Lazy model that can not commit yet

In the case of a Lazy model where we are forced to Abort

If we have already delayed, go ahead and finalize commit in memory

Move instruction pointer to next instruction

4.6.3.19 void transactionContext::createStall (thread_ptr *pthread*, int *stallLength*) `[private]`

Stalls a thread for a given period.

Parameters:

pthread SESC thread pointer

stallLength Length of stall

4.6.3.20 void transactionContext::stallInstruction (thread_ptr *pthread*, icode_ptr *picode*, int *stallLength*) `[private]`

Stalls the GCM for a given period.

Parameters:

pthread SESC thread pointer

picode Instruction code

stallLength Length of stall

This method tells the GCM to stall for the next stallLength cycles as well as creates a duplicate of the current instruction so that we will try it again after the stall period

Note:

We are going to have to change the Next pointer of the real instruction since it will automatically incremented by the default instruction handler. To do this, we need to create an identical copy of the instruction, and use this

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

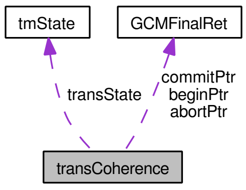
- simulators/sesc/SESC-Gold/src/libtrans/**transContext.h**
- simulators/sesc/SESC-Gold/src/libtrans/**transContext.cpp**

4.7 transCoherence Class Reference

TM Coherency Manager.

`#include <transCoherence.h>`

Collaboration diagram for transCoherence:



Public Member Functions

- struct **GCMFinalRet abortEE** (thread_ptr pthread, int tid)

  eager eager abort

- struct **GCMFinalRet abortLL** (thread_ptr pthread, int tid)

  lazy lazy abort

- struct **GCMFinalRet beginEE** (int pid, icode_ptr picode)

  eager eager begin

- struct **GCMFinalRet beginLL** (int pid, icode_ptr picode)

  lazy lazy begin

- bool **checkAbort** (int pid, int tid)

  check to see if thread has been ordered to abort

- struct **GCMFinalRet commitEE** (int pid, int tid)

  eager eager commit

- struct **GCMFinalRet commitLL** (int pid, int tid)

  lazy lazy commit

- **GCMRet readEE** (int pid, int tid, RAddr raddr)

  eager eager read

- **GCMRet readLL** (int pid, int tid, RAddr raddr)

  lazy lazy read

- **transCoherence** (FILE ∗out, int conflicts, int versioning, int cacheLineSize)

  Constructor.

- **transCoherence** ()

  Global Coherence Module.

- **GCMRet writeEE** (int pid, int tid, RAddr raddr)

  eager eager write

- **GCMRet writeLL** (int pid, int tid, RAddr raddr)

  lazy lazy write

Private Member Functions

- struct **cacheState newReadState** (int pid)

  Create new cache state reference with Read bit set.

- struct **cacheState newWriteState** (int pid)

  Create new cache state reference with Write bit set.

Private Attributes

- int **currentCommitter**

  PID of the currently committing processor.

- map< RAddr, **cacheState** > **permCache**

  The cache ownership.

- long long int **utid**

  Unique Global Transaction ID.

4.7.1 Detailed Description

TM Coherency Manager.

Coordinates the entire coherency of the transactional memory system. Read/Write/Abort/Commit/Begin must all be provided and linked to functional pointers at runtime to determine EE/LL/etc.

4.7.2 Member Function Documentation

4.7.2.1 struct GCMFinalRet transCoherence::abortEE (thread_ptr *pthread*, int *tid*) `[read]`

eager eager abort

Parameters:

pthread SESC pointer to thread

tid Thread ID

Returns:

Final coherency status

We can't just decriment because we should be going back to the original begin, so tmDepth[pid] = 0

4.7.2.2 struct GCMFinalRet transCoherence::abortLL (thread_ptr *pthread*, int *tid*) `[read]`

lazy lazy abort

Parameters:

pthread SESC thread pointer

tid Thread ID

Returns:

Final coherency status

We can't just decriment because we should be going back to the original begin, so tmDepth[pid] = 0

Write set size doesn't matter for Lazy/Lazy abort

4.7.2.3 GCMFinalRet transCoherence::beginEE (int *pid*, icode_ptr *picode*) `[read]`

eager eager begin

Parameters:

pid Process ID

picode Instruction code

Returns:

Final coherency status

Subsume all nested transactions for now

This is a subsumed begin, set BCFlag = 2

If we had just aborted, we need to now invalidate all the memory addresses we touched

If we just finished an abort, its time to backoff

Pass whether this is the begining of an aborted replay back to the context

Replay

4.7.2.4 GCMFinalRet transCoherence::beginLL (int *pid*, icode_ptr *picode*) `[read]`

lazy lazy begin

Parameters:

pid Process ID

picode SESC icode pointer

Returns:

Final coherency status

Subsume all nested transactions for now

This begin is subsumed, set the BCFlag to 2

If we had just aborted, we need to now invalidate all the memory addresses we touched

Pass whether this is the begining of an aborted replay back to the context

Replay

4.7.2.5 bool transCoherence::checkAbort (int *pid*, int *tid*)

check to see if thread has been ordered to abort

Parameters:

pid Process ID
tid Thread ID

Returns:

Abort?

4.7.2.6 struct GCMFinalRet transCoherence::commitEE (int *pid*, int *tid*) `[read]`

eager eager commit

Parameters:

pid Process ID
tid Thread ID

Returns:

Final coherency status

Set the default BCFlag to 0, since the only other option for Commit is subsumed 2

This commit is subsumed, set the BCFlag to 2

If we have already stalled for the commit, our state will be COMMITTING, Complete Commit

Register Commit in Report

4.7.2.7 struct GCMFinalRet transCoherence::commitLL (int *pid*, int *tid*) `[read]`

lazy lazy commit

Parameters:

pid Process ID

tid Thread ID

Returns:

Final coherency status

Set BCFlag default to 0, since only other option is subsumed BCFlag = 2

If we have been forced to ABORT

This is a subsumed commit, set BCFlag = 2

If we have already stalled for the commit, our state will be COMMITTING, Complete Commit

Register Commit in Report

If we have written to this address, we must abort everyone who read/wrote to it

Increase our write set

Abort all who wrote to this

Abort all who read from this

Allow other transaction to commit again

Register Commit in Report

Stop other transactions from being able to commit

4.7.2.8 struct cacheState transCoherence::newReadState (int *pid*) `[read, private]`

Create new cache state reference with Read bit set.

Parameters:

pid Process ID

Returns:

Cache state

4.7.2.9 struct cacheState transCoherence::newWriteState (int *pid*) `[read, private]`

Create new cache state reference with Write bit set.

Parameters:

pid Process ID

Returns:

Cache state

4.7.2.10 GCMRet transCoherence::readEE (int *pid*, int *tid*, RAddr *raddr*)

eager eager read

Parameters:

pid Process ID

tid Thread ID

raddr Real address

Returns:

Coherency status

If the cache line has been instantiated in our Map

We haven't, so create a new one

4.7.2.11 GCMRet transCoherence::readLL (int *pid*, int *tid*, RAddr *raddr*)

lazy lazy read

Parameters:

pid Process ID

tid Thread ID

raddr Real address

Returns:

Coherency status

If we have been forced to ABORT

If the cache line has been instantiated in our Map

We haven't, so create a new one

4.7.2.12 GCMRet transCoherence::writeEE (int *pid*, int *tid*, RAddr *raddr*)

eager eager write

Parameters:

pid Process ID

tid Thread ID

raddr Real address

Returns:

Coherency status

If the cache line has been instantiated in our Map

If there is more than one reader, or there is a single reader who happens not to be us

Grab the first reader than isn't us

Take our timestamp as well as the readers

If the process that is going to nack us is older than us, and we have cycle flag set, abort

If we are older than the guy we're nacking on, then set her cycle flag to indicate possible deadlock

Grab the first reader than isn't us

We haven't, so create a new one

4.7.2.13 GCMRet transCoherence::writeLL (int *pid*, int *tid*, RAddr *raddr*)

lazy lazy write

Parameters:

pid Process ID

tid Thread ID

raddr Real address

Returns:

Coherency status

If we have been forced to ABORT

If the cache line has been instantiated in our Map

We haven't, so create a new one

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

- simulators/sesc/SESC-Gold/src/libtrans/**transCoherence.h**
- simulators/sesc/SESC-Gold/src/libtrans/**transCoherence.cpp**

4.8 transRef Struct Reference

used for transMemRef option

`#include <transReport.h>`

4.8.1 Detailed Description

used for transMemRef option

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

- simulators/sesc/SESC-Gold/src/libtrans/**transReport.h**

4.9 transReport Class Reference

Report Module.

`#include <transReport.h>`

Public Member Functions

- void **beginTMStats** (**INSTCOUNT** insts)

  Sets the begining Instruction/Cycle count of transactional workloads.

- void **registerBegin** (**ID** utid, int pid, int tid, RAddr PC, **TIMESTAMP** timestamp)

  register begin (at fetch)

- void **registerCommit** (**ID** utid, int pid, int tid, **TIMESTAMP** timestamp)

  register abort (at fetch)

- void **registerLoad** (**ID** utid, RAddr beginPC, int pid, int tid, RAddr raddr, RAddr caddr, **TIMESTAMP** timestamp)

  register load (fetch)

- void **registerStore** (**ID** utid, RAddr beginPC, int pid, int tid, RAddr raddr, RAddr caddr, **TIMESTAMP** timestamp)

  register store (fetch)

- void **registerTransInst** (int pid, transInstType type)

  increment transactional instruction count

- void **registerTransInstAbort** (int pid, transInstType type)

  increment transactional instruciton count (for aborts)

- void **reportAbort** (**ID** utid, int pid, int tid, int nackPid, RAddr raddr, RAddr caddr, **TIMESTAMP** myTimeStamp, **TIMESTAMP** nackTimestamp)

  report abort

- void **reportBegin** (int pid, int cpu)

  report begin

- void **reportCommit** (int pid)

  report commit

- void **reportLoad** (int pid)

  report load (retire)

- void **reportNackCommit** (**ID** utid, int pid, int tid, int nackPid, **TIMESTAMP** my-Timestamp, **TIMESTAMP** nackTimestamp)

  report nack on commit

- void **reportNackCommitFN** (**ID** utid, int pid, int tid, **TIMESTAMP** begin_-timestamp) *report nack resolved*

- void **reportNackLoad** (**ID** utid, int pid, int tid, int nackPid, RAddr raddr, RAddr caddr, **TIMESTAMP** myTimestamp, **TIMESTAMP** nackTimestamp)

  report nack on load

- void **reportNackStore** (**ID** utid, int pid, int tid, int nackPid, RAddr raddr, RAddr caddr, **TIMESTAMP** myTimestamp, **TIMESTAMP** nackTimestamp)

  report nack on store

- void **reportStore** (int pid)

  report store (retire)

- std::set< RAddr > **return_globalReadSet** (void)

  get global read set

- std::set< RAddr > **return_globalWriteSet** (void)

  get global write set

- void **summaryAbort** (int pid, **INSTCOUNT** instCount)

  Handles aborts for summary.

- void **summaryBegin** (int pid, **TIMESTAMP** timestamp)

  Handles begins for summary.

- void **summaryCommit** (int pid, **INSTCOUNT** instCount, **TIMESTAMP** timestamp)

  Handles commits for summary.

- void **summaryComplete** ()

  global report output final results

- void **summaryLoad** (int pid, RAddr addr)

  global report load

- void **summaryNackBegin** (int pid, **TIMESTAMP** timestamp)

  Handles NACK begins for summary.

- void **summaryNackFinish** (int pid, **TIMESTAMP** timestamp)

  global report nack finish

- void **summaryStore** (int pid, RAddr addr)

  global report store

- void **transactionalAbort** (**ID** utid, **INSTCOUNT** instCount)

  transactional report abort

- void **transactionalBegin** (**ID** utid, int tid, int pid, RAddr PC, int cpu, **TIMESTAMP** timestamp)

  transactional report begin

- void **transactionalCommit** (**ID** utid, **INSTCOUNT** instCount, **TIMESTAMP** timestamp, **INSTCOUNT** fpOps)

  transactional report commit

- void **transactionalComplete** ()

  transactional report cleanup

- void **transactionalCompleteSummary** ()

  transactional report final summary output

- void **transactionalLoad** (**ID** utid, RAddr addr)

  transactional report load

- void **transactionalNackBegin** (**ID** utid, int tid, int confPid, RAddr raddr, **TIMESTAMP** timestamp)

  transactional report nack begin

- void **transactionalNackFinish** (**ID** utid, **TIMESTAMP** timestamp)

  transactional report nack finish

- void **transactionalStore** (**ID** utid, RAddr addr)

  transactional report store

- std::list< list< RAddr > > **transMemRef_getLoadLists** (RAddr PC)

  Returns the list of lists of memory references for a given PC address.

- std::map< RAddr, list< list< RAddr > > > **transMemRef_getLoadMap** ()

  Returns the transMemHistoryLoads map (L).

- std::list< list< RAddr > > **transMemRef_getStoreLists** (RAddr PC)

  Returns the list of lists of memory references for a given PC address.

- std::map< RAddr, list< list< RAddr > > > **transMemRef_getStoreMap** ()

  Returns the transMemHistoryStores map (S).

- void **transMemRef_printResults** ()

  Prints out the entire map. Useful for validation.

- std::list< list< RAddr > > * **transMemRef_ref_getLoadLists** (RAddr PC)

  transMemRef get load addresses for PC

- std::list< list< RAddr > > * **transMemRef_ref_getStoreLists** (RAddr PC)

  transMemRef get store list for PC

- **transReport** (const char *reportFileName)

  Constructor.

Private Member Functions

- void **transMemRef_newBegin** (int pid, RAddr PC)
  transMemRef store function
- void **transMemRef_newCommit** (int pid)
  transMemRef commit
- void **transMemRef_newLoad** (RAddr PC, RAddr mem)
  transMemRef load
- void **transMemRef_newStore** (RAddr PC, RAddr mem)
  transMemRef store

4.9.1 Detailed Description

Report Module.

This module is used to calculate the output statistics. It keeps track of cycle counts, instruction counts, etc and writes them to an output file based on the level of granularity (detailed, transactional, global) the user wants

4.9.2 Constructor & Destructor Documentation

4.9.2.1 transReport::transReport (const char ∗ *reportFileName*)

Constructor.

Parameters:

reportFileName

4.9.3 Member Function Documentation

4.9.3.1 void transReport::beginTMStats (INSTCOUNT *insts*)

Sets the begining Instruction/Cycle count of transactional workloads.

Parameters:

insts instruction count

4.9.3.2 void transReport::registerBegin (ID *utid*, int *pid*, int *tid*, RAddr *PC*, TIMESTAMP *begin_timestamp*)

register begin (at fetch)

Parameters:

utid Unique Tx ID

pid Process ID
tid ThreadID
PC Program counter
begin_timestamp Cycle when Tx begins

Reset the transactional Abort Instruction Counter

4.9.3.3 void transReport::registerCommit (ID *utid*, int *pid*, int *tid*, TIMESTAMP *begin_timestamp*)

register abort (at fetch)

Parameters:

utid Unique Tx ID
pid Process ID
tid ThreadID
begin_timestamp Cycle when Tx begins

4.9.3.4 void transReport::registerLoad (ID *utid*, RAddr *beginPC*, int *pid*, int *tid*, RAddr *raddr*, RAddr *caddr*, TIMESTAMP *begin_timestamp*)

register load (fetch)

Parameters:

utid Unique Tx ID
beginPC PC when Tx begins
pid Process ID
tid ThreadID
raddr Real address
caddr Cache address
begin_timestamp Cycle when Tx begins

4.9.3.5 void transReport::registerStore (ID *utid*, RAddr *beginPC*, int *pid*, int *tid*, RAddr *raddr*, RAddr *caddr*, TIMESTAMP *begin_timestamp*)

register store (fetch)

Parameters:

utid Unique Tx ID
beginPC PC when Tx begins
pid Process ID
tid ThreadID
raddr Real address
caddr Cache address
begin_timestamp Cycle when Tx begins

4.9.3.6 void transReport::registerTransInst (int *pid*, transInstType *type*)

increment transactional instruction count

Parameters:

pid Process ID

type Transactional instruction type

4.9.3.7 void transReport::registerTransInstAbort (int *pid*, transInstType *type*)

increment transactional instruciton count (for aborts)

Parameters:

pid Process ID

type Transactional instruction type

Note:

This count is very similiar to the previous, but happens at the fetch stage so that we can get instruction counts of the Aborted transactions

4.9.3.8 void transReport::reportAbort (ID *utid*, int *pid*, int *tid*, int *nackPid*, RAddr *raddr*, RAddr *caddr*, TIMESTAMP *myTimestamp*, TIMESTAMP *nackTimestamp*)

report abort

Parameters:

utid Unique Tx ID

pid Process ID

tid ThreadID

nackPid Process ID of nacking thread

raddr Real address

caddr Cache address

myTimestamp my timestamp

nackTimestamp timestamp of aborter

Handle the case for the Eager approaches

Handle the case for Lazy approaches

4.9.3.9 void transReport::reportBegin (int *pid*, int *cpu*)

report begin

Note:

CPU IS SAVED FOR THE ENTIRE TRANSACTION, THUS THIS ASSUMES NO MIGRATION OF IN-FLIGHT TRANSACTIONS. OR, TRANSACTION WILL BE PRINTED IN STATISTICS AS PART OF THE CPU IT BEGAN ON.

Parameters:

pid Process ID

cpu CPU ID

Right now we are subsuming nest transactions, so we need to check if this is a nested begin

Reset the transactional Instruction Counter

4.9.3.10 void transReport::reportCommit (int *pid*)

report commit

Parameters:

pid Process ID

Right now we are subsuming nest transactions, so we need to check if this is a nested begin

Always fflush on a Commit

4.9.3.11 void transReport::reportLoad (int *pid*)

report load (retire)

Parameters:

pid Process ID

4.9.3.12 void transReport::reportNackCommit (ID *utid*, int *pid*, int *tid*, int *nackPid*, TIMESTAMP *myTimestamp*, TIMESTAMP *nackTimestamp*)

report nack on commit

Parameters:

utid Unique Tx ID

pid Process ID

tid ThreadID

nackPid Process ID of nacking thread

myTimestamp my timestamp

nackTimestamp timestamp of nacker

4.9.3.13 void transReport::reportNackCommitFN (ID *utid*, int *pid*, int *tid*, TIMESTAMP *begin_timestamp*)

report nack resolved

Parameters:

utid Unique Tx ID

pid Process ID

tid ThreadID

begin_timestamp Cycle when Tx begins

Note:

This can not be embedded into the commit function as it is in memory functions for lazy approaches since that would also include the commit time in the nack delay.

If we are able to commit in a Lazy approach and we were stalling, print NKFN

4.9.3.14 void transReport::reportNackLoad (ID *utid*, int *pid*, int *tid*, int *nackPid*, RAddr *raddr*, RAddr *caddr*, TIMESTAMP *myTimestamp*, TIMESTAMP *nackTimestamp*)

report nack on load

Parameters:

utid Unique Tx ID

pid Process ID

tid ThreadID

nackPid Process ID of nacking thread

raddr Real address

caddr cache address

myTimestamp my timestamp

nackTimestamp timestamp of nacker

This will only print out the NACK if it is a new, unique NACK. Note the timestamp must not be equal to ((~0ULL) - 1024) because this 1indicates that a transaction is in the process of ABORTING, but has yet to complete its ABORT.

4.9.3.15 void transReport::reportNackStore (ID *utid*, int *pid*, int *tid*, int *nackPid*, RAddr *raddr*, RAddr *caddr*, TIMESTAMP *myTimestamp*, TIMESTAMP *nackTimestamp*)

report nack on store

Parameters:

utid Unique Tx ID

pid Process ID

tid ThreadID

nackPid Process ID of nacking thread

raddr Real address

caddr cache adress

myTimestamp my timestamp

nackTimestamp timestamp of nacker

This will only print out the NACK if it is a new, unique NACK. Note the timestamp must not be equal to ((~0ULL) - 1024) because this indicates that a transaction is in the process of ABORTING, but has yet to complete its ABORT. You can disable this check with printAllNacks.

4.9.3.16 void transReport::reportStore (int *pid*)

report store (retire)

Parameters:

pid Process ID

4.9.3.17 std::set< RAddr > transReport::return_globalReadSet (void)

get global read set

Returns:

4.9.3.18 std::set< RAddr > transReport::return_globalWriteSet (void)

get global write set

Returns:

4.9.3.19 void transReport::summaryAbort (int *pid*, INSTCOUNT *instCount*)

Handles aborts for summary.

Parameters:

pid Process ID

instCount Total instructions

Used For: Global Summary Statistics Called on an AB!! to increment the abort count, as well as incrementing the instruction counts for aborted instructions. Does not handle cycle information, that is handled in Begin for timing accurate results.

4.9.3.20 void transReport::summaryBegin (int *pid*, TIMESTAMP *timestamp*)

Handles begins for summary.

Parameters:

pid Process ID

timestamp Internal time

Used For: Global Summary Statistics Called on a Begin to store the current timestamp as well as clear all of the counters.

Note:

If our summaryTransFlag is set, this means that we were currently in a transaction and received another Begin. This implies the previous transaction aborted, thus we can take abort cycle counts.

This does NOT increment the abort count, because to get the instruction counts we have an explicit Abort function that increments the count.

Also Note: This will have to be modified if we eventually support nested transactions.

4.9.3.21 void transReport::summaryCommit (int *pid*, INSTCOUNT *instCount*, TIMESTAMP *timestamp*)

Handles commits for summary.

Parameters:

pid Process ID

instCount Total instructions

timestamp Internal time

Used For: Global Summary Statistics Called on a Commit to increment the Commit count as well as the instruction counts for commits for that transaction. Also calculates the length of the transaction in cycles. Note, we also only count read/write set counts here because aborted transactions don't give an accurate view of the total read/write set size (since they do not complete).

4.9.3.22 void transReport::summaryComplete ()

global report output final results

Used For: Global Summary Statistics Printed at the end of execution to summarize global transactional results.

Global statistics not reliable for parsing between useful/aborted nacks, just total

Global statistics not reliable for parsing between useful/aborted nacks, just total

4.9.3.23 void transReport::summaryLoad (int *pid*, RAddr *addr*)

global report load

Parameters:

pid Process ID
addr Real address

Used For: Global Summary Statistics Called on a load, adds the address to a write set for the transaction and increments the load count for the transaction.

4.9.3.24 void transReport::summaryNackBegin (int *pid*, TIMESTAMP *timestamp*)

Handles NACK begins for summary.

Parameters:

pid Process ID
timestamp Internal time

Used For: Global Summary Statistics Called at the begining of a NK so that we can record the start time and thus calulate the total length of the stall on the NKFN. Also increments the Nack Count for that transaction.

4.9.3.25 void transReport::summaryNackFinish (int *pid*, TIMESTAMP *timestamp*)

global report nack finish

Parameters:

pid Process ID
timestamp Internal time

Used For: Global Summary Statistics Called on a NKFN to calculate the length of the Nack stall defaults to a minimum value of 1.

4.9.3.26 void transReport::summaryStore (int *pid*, RAddr *addr*)

global report store

Parameters:

pid Process ID
addr Real address

Used For: Global Summary Statistics Called on a store, adds the address to a write set and increments the store count for the transaction.

4.9.3.27 void transReport::transactionalAbort (ID *utid*, INSTCOUNT *instCount*)

transactional report abort

Parameters:

utid Unique Tx ID
instCount Total instructions

4.9.3.28 void transReport::transactionalBegin (ID *utid*, int *tid*, int *pid*, RAddr *PC*, int *cpu*, TIMESTAMP *timestamp*)

transactional report begin

Parameters:

utid Unique Tx ID

tid ThreadID

pid Process ID

PC Program counter

cpu CPU ID

timestamp Internal time

We're starting a new transaction on the same process without a commit Last must have aborted (since we already filter out subsumed)

4.9.3.29 void transReport::transactionalCommit (ID *utid*, INSTCOUNT *instCount*, TIMESTAMP *timestamp*, INSTCOUNT *fpOps*)

transactional report commit

Parameters:

utid Unique Tx ID

instCount Total instructions

timestamp Internal time

fpOps Number of FP operations

4.9.3.30 void transReport::transactionalCompleteSummary ()

transactional report final summary output

THIS LINE DEFINES WHAT WE CONSIDER THE "PID"

4.9.3.31 void transReport::transactionalLoad (ID *utid*, RAddr *addr*)

transactional report load

Parameters:

utid Unique Tx ID

addr Real address

4.9.3.32 void transReport::transactionalNackBegin (ID *utid*, int *tid*, int *confPid*, RAddr *raddr*, TIMESTAMP *timestamp*)

transactional report nack begin

Parameters:

utid Unique Tx ID

tid ThreadID

confPid Conflicting process ID

raddr Real address

timestamp Internal time

Since it's possible for a NK to appear before the Begin in the commit stage, we need to ensure that there is an entry in the transDataReport file before we add the conflict If there isn't one, we will create a very short temporary one.

4.9.3.33 void transReport::transactionalNackFinish (ID *utid*, TIMESTAMP *timestamp*)

transactional report nack finish

Parameters:

utid Unique Tx ID

timestamp Internal time

4.9.3.34 void transReport::transactionalStore (ID *utid*, RAddr *addr*)

transactional report store

Parameters:

utid Unique Tx ID

addr Real address

4.9.3.35 std::list< list< RAddr > > transReport::transMemRef_getLoadLists (RAddr *PC*)

Returns the list of lists of memory references for a given PC address.

Parameters:

PC Program counter

Returns:

History of memory operations (L)

4.9.3.36 map< RAddr, list< list< RAddr > > > transReport::transMemRef_-getLoadMap ()

Returns the transMemHistoryLoads map (L).

Returns:

History of memory operations (L)

4.9.3.37 std::list< list< RAddr > > transReport::transMemRef_getStoreLists (RAddr *PC*)

Returns the list of lists of memory references for a given PC address.

Parameters:

PC Program counter

Returns:

History of memory operations (S)

4.9.3.38 map< RAddr, list< list< RAddr > > > transReport::transMemRef_-getStoreMap ()

Returns the transMemHistoryStores map (S).

Returns:

History of memory operations (S)

4.9.3.39 void transReport::transMemRef_newBegin (int *pid*, RAddr *PC*) `[private]`

transMemRef store function

Parameters:

pid Process ID

PC Program counter

Function called on new Begin instruction. First checks to see if the map has an entry for this PC. If it does not, it adds a new list of lists of RAddrs for that PC. Otherwise it simply adds a new list of RAddrs into the current list at that PC. Note: It must also ensure the previous transaction wasn't aborted. If it was, it simply clears the RAddrs from the list and continues.

Ensure that we have experienced a commit since the last begin for this pid (this ensures the previous transaction on this pid wasn't aborted).

Set the "running" state to true so that we can detect an abort

First we will take care of LOADs

If the PC isn't currently in our map, add an entry for it

Otherwise, simply create a new list and add it to the current

Now we take care of the STOREs list

If the PC isn't currently in our map, add an entry for it

Otherwise, simply create a new list and add it to the current

If we have not encountered a commit, then the previous transaction was aborted. Since we do not want to keep track of aborted transactions simply clear out the current list of RAddrs and continue

4.9.3.40 void transReport::transMemRef_newCommit (int *pid*) `[private]`

transMemRef commit

Parameters:

pid Process ID

Since we only want to add to our map memory addresses for completed transactions, this is used to clear a flag that allows us to detect aborted transactions. Note subsumed transactions are detected elsewhere, so they are not an issue.

4.9.3.41 void transReport::transMemRef_newLoad (RAddr *PC*, RAddr *mem*) `[private]`

transMemRef load

Parameters:

PC Program counter

mem Real address

On a load we simply push the memory address onto the current list

4.9.3.42 void transReport::transMemRef_newStore (RAddr *PC*, RAddr *mem*) `[private]`

transMemRef store

Parameters:

PC Program counter

mem Real address

On a Store we simply push the memory address onto the current list

4.9.3.43 std::list< list< RAddr > > * transReport::transMemRef_ref_getLoadLists (RAddr *PC*)

transMemRef get load addresses for PC

Parameters:

PC Program counter

Returns:

4.9.3.44 std::list< list< RAddr > > * transReport::transMemRef_ref_getStoreLists (RAddr *PC*)

transMemRef get store list for PC

Parameters:

PC Program counter

Returns:

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

- simulators/sesc/SESC-Gold/src/libtrans/**transReport.h**
- simulators/sesc/SESC-Gold/src/libtrans/**transReport.cpp**

Chapter 5

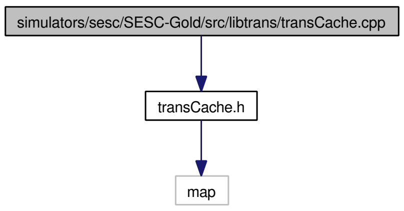
File Documentation

5.1 simulators/sesc/SESC-Gold/src/libtrans/transCache.cpp File Reference

This is the implementation for the TM cache manager.

`#include "transCache.h"`

Include dependency graph for transCache.cpp:



5.1.1 Detailed Description

This is the implementation for the TM cache manager.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.1.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.1.3 DESCRIPTION

C++ Implementation: **transactionCache** (p. 11)

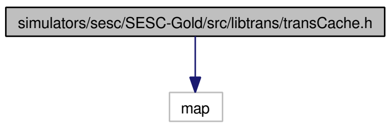
5.2 simulators/sesc/SESC-Gold/src/libtrans/transCache.h File Reference

This is the interface for the TM cache manager.

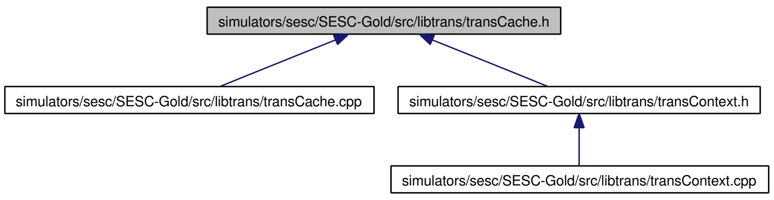
`#include <map>`

`#include "SescConf.h"`

Include dependency graph for transCache.h:



This graph shows which files directly or indirectly include this file:


simulators/sesc/SESC-Gold/src/libtrans/transCache.h
simulators/sesc/SESC-Gold/src/libtrans/transCache.cpp
simulators/sesc/SESC-Gold/src/libtrans/transContext.h
simulators/sesc/SESC-Gold/src/libtrans/transContext.cpp

Data Structures

- class **transactionCache**

  Transaction Cache.

Defines

- #define **SWAP_SHORT**(X) (((((unsigned short)X)& 0xff00) >> 8) | ((((unsigned short)X)& 0x00ff) << 8))
- #define **SWAP_WORD**(X)

5.2.1 Detailed Description

This is the interface for the TM cache manager.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.2.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.2.3 DESCRIPTION

C++ Interface: **transactionCache** (p. 11) Functional cache used to store transactional data. At the functional level, all models operate similiar to L/L with an infinite size temporary cache.

5.2.4 Define Documentation

5.2.4.1 #define SWAP_SHORT(X) (((((unsigned short)X)& 0xff00) >> 8) | ((((unsigned short)X)& 0x00ff) << 8))

Endianness (short)

5.2.4.2 #define SWAP_WORD(X)

Value:

```
(((((unsigned int)(X)) >> 24) & 0x000000ff) | \
                               ((((unsigned int)(X)) >>  8) & 0x0000ff00) | \
                                                        ((((unsigned int)(X)) <<  8) & 0x00ff0000) | \
                                                        ((((unsigned int)(X)) << 24) & 0xff000000))
```

Endianness (word)

5.3 simulators/sesc/SESC-Gold/src/libtrans/transCoherence.cpp File Reference

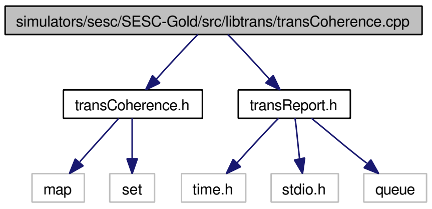
This is the implementation for the global coherence module.

`#include "transCoherence.h"`

`#include "ThreadContext.h"`

`#include "transReport.h"`

Include dependency graph for transCoherence.cpp:



5.3.1 Detailed Description

This is the implementation for the global coherence module.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.3.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.3.3 DESCRIPTION

C++ Implementation: **transCoherence** (p. 23)

5.4 simulators/sesc/SESC-Gold/src/libtrans/transCoherence.h File Reference

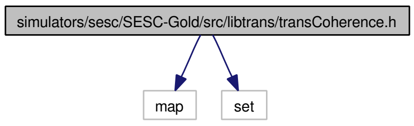
This is the interface for the global coherence module.

`#include <map>`

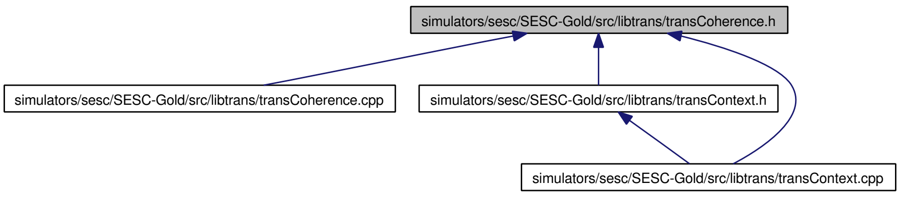
`#include <set>`

`#include "icode.h"`

Include dependency graph for transCoherence.h:



This graph shows which files directly or indirectly include this file:


simulators/sesc/SESC-Gold/src/libtrans/transCoherence.h
simulators/sesc/SESC-Gold/src/libtrans/transCoherence.cpp
simulators/sesc/SESC-Gold/src/libtrans/transContext.h
simulators/sesc/SESC-Gold/src/libtrans/transContext.cpp

Data Structures

- struct **cacheState**

  Cache Line State Container (R/W).

- struct **GCMFinalRet**

  Structure used to store result of coherence request.

- struct **tmState**

  TM State Container.

- class **transCoherence**

  TM Coherency Manager.

Enumerations

- enum **condition**
- enum **GCMRet**
- enum **perState**

5.4.1 Detailed Description

This is the interface for the global coherence module.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.4.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.4.3 DESCRIPTION

C++ Interface: **transCoherence** (p. 23)

This object provides the functional coherence.

Note:

Commits/Aborts have two phases. The first induces the stall, the second is where all of the memory addresses are freed.

5.4.4 Enumeration Type Documentation

5.4.4.1 enum condition

Transaction conditions.

5.4.4.2 enum GCMRet

Global coherency module return values.

5.4.4.3 enum perState

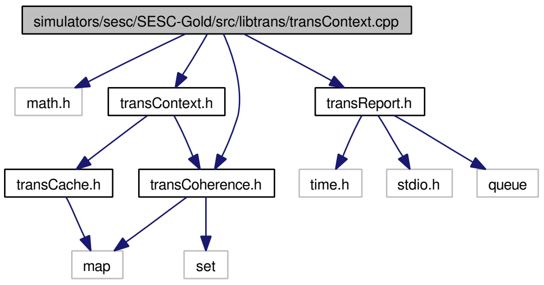
Cache line state values.

5.5 simulators/sesc/SESC-Gold/src/libtrans/transContext.cpp File Reference

This is the implementation for the transaction context module.

```
#include <math.h>
#include "transContext.h"
#include "transReport.h"
#include "ThreadContext.h"
#include "transCoherence.h"
#include "opcodes.h"
```

Include dependency graph for transContext.cpp:



5.5.1 Detailed Description

This is the implementation for the transaction context module.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.5.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.5.3 DESCRIPTION

C++ Implementation: **transactionContext** (p. 15)

5.6 simulators/sesc/SESC-Gold/src/libtrans/transContext.h File Reference

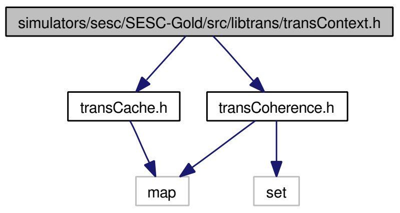
This is the interface for the transaction context module.

`#include "transCache.h"`

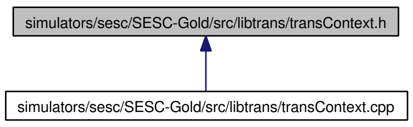
`#include "transCoherence.h"`

`#include "icode.h"`

Include dependency graph for transContext.h:



This graph shows which files directly or indirectly include this file:


simulators/sesc/SESC-Gold/src/libtrans/transContext.h
simulators/sesc/SESC-Gold/src/libtrans/transContext.cpp

Data Structures

- class **transactionContext**

  transactional context

5.6.1 Detailed Description

This is the interface for the transaction context module.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.6.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.6.3 DESCRIPTION

C++ Interface: **transactionContext** (p. 15)

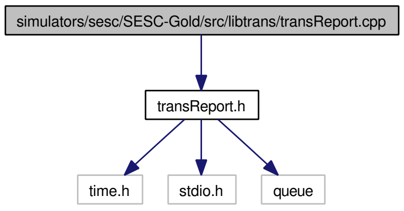
Full description here.

5.7 simulators/sesc/SESC-Gold/src/libtrans/transReport.cpp File Reference

This is the implementation for the transaction reporting module.

`#include "transReport.h"`

Include dependency graph for transReport.cpp:



5.7.1 Detailed Description

This is the implementation for the transaction reporting module.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.7.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.7.3 DESCRIPTION

C++ Implementation: **transReport** (p. 31)

5.8 simulators/sesc/SESC-Gold/src/libtrans/transReport.h File Reference

This is the interface for the transaction reporting module.

`#include <time.h>`

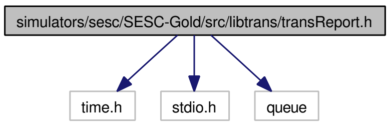
`#include <stdio.h>`

`#include <queue>`

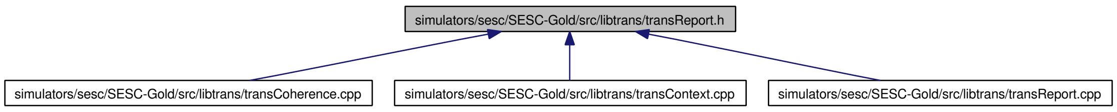
`#include "OSSim.h"`

`#include "ExecutionFlow.h"`

Include dependency graph for transReport.h:



This graph shows which files directly or indirectly include this file:


simulators/sesc/SESC-Gold/src/libtrans/transReport.h
simulators/sesc/SESC-Gold/src/libtrans/transCoherence.cpp
simulators/sesc/SESC-Gold/src/libtrans/transContext.cpp
simulators/sesc/SESC-Gold/src/libtrans/transReport.cpp

Data Structures

- struct **memRef**

  memory reference structure used for transMemRef

- struct **transRef**

  used for transMemRef option

- class **transReport**

  Report Module.

Typedefs

- typedef unsigned long long **ID**
- typedef unsigned long long **INSTCOUNT**
- typedef unsigned long long **TIMESTAMP**

5.8.1 Detailed Description

This is the interface for the transaction reporting module.

Author:

jpoe <>, (C) 2008, 2009

Date:

09/19/08

5.8.2 LICENSE

Copyright: See COPYING file that comes with this distribution

5.8.3 DESCRIPTION

C++ Interface: **transReport** (p. 31)

This file is responsible for creating the transactional trace file that is used to extract useful statistics from the simulation. Note for most operations there is a "register" and a "report" version. The purpose of the register is to add the operation to a queue for later print, and then the report function is called at the commit stage to actually print the operation so that we have accurate cycle counts.

5.8.4 Typedef Documentation

5.8.4.1 ID

unsigned long long.

5.8.4.2 INSTCOUNT

unsigned long long

5.8.4.3 TIMESTAMP

unsigned long long

Index