| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [FixedModeFlow/src](http://docs.google.com/index.html) - SlidingWindow.cpp (source / [functions](http://docs.google.com/SlidingWindow.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 215 | 268 | 80.2 % | | Date: | 2012-03-23 |  | Functions: | 19 | 19 | 100.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 68 | 132 | 51.5 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #include <algorithm>  2 : : #include <strings.h>  3 : :   4 : : #include "ConverterParams.h"  5 : : #include "FlagField.h"  6 : : #include "HealthCheckCounter.h"  7 : : #include "Memory.h"  8 : : #include "SlidingWindow.h"  9 : : #include "SlidingWindowPositions.h"  10 : : #include "SFTable.h"  11 : : #include "ThreadManager.h"  12 : : #include "VSBDataProcessor.h"  13 : : #include "WarningHandler.h"  14 : :   15 : : #include "print.h"  16 : : #include "SDDB.h"  17 : :   18 : : namespace nft {  19 : :   20 : : SlidingWindow::SlidingWindow( ThreadID inThreadID,  21 : : SFTable::Pointer inSFTable )  22 : 1048 : :Thread(),  23 : : mPositions(),  24 : : mSFTable(inSFTable),  25 : : mCurrentStartIndex(),  26 : : mCurrentProcessedIndex(),  27 : : mCurrentEndIndex(),  28 : : mCellProcessors(),  29 : : mCellScheduler( mCellProcessors ),  30 : : mOutputGenerator(),  31 : : mMutex(),  32 : : mWaitCondition(),  33 : : mNumActiveProcessors(),  34 : : mStop(),  35 : : mThreadDataSizes(),  36 : : mBBox(),  37 : 2096 : mDataProc() {  38 : 1048 : setThreadID( inThreadID );  39 : : }  40 : :   41 : 2096 : SlidingWindow::~SlidingWindow() {  42 : 1048 : mCellProcessors.clear();  43 [ + - ][ # # ]: 1048 : delete mPositions;  [ # # ]  44 [ + - ][ # # ]: 1048 : delete mDataProc;  [ # # ]  45 : 1048 : }  46 : :   47 : : void  48 : 1048 : SlidingWindow::generateWindowPositions() throw(DPL::Error \*) {  49 : 1048 : mPositions = new(std::nothrow) SlidingWindowPositions();  50 : : //#ifdef ERROR\_SEED //<<Test\_Id:Sliding\_Window\_mPositions\_Set\_To\_NULL>>  51 : : // mPositions=NULL;  52 : : //#endif  53 [ - + ]: 1048 : if( NULL == mPositions )  54 : : {  55 : 0 : int iErrNo = errno; //System error no  56 : 0 : char systemErrorMsg[BUFSIZ];  57 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  58 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  59 : : ErrorHandler::eErrMemoryAllocation,  60 : : pErr->getErrorMsg(  61 : : ErrorHandler::eErrMemoryAllocation,  62 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  63 : : }  64 : 1048 : ConverterParams \*params = ConverterParams::instance();  65 : 1048 : SFLayer::Pointer layers = mSFTable->getSFLayers();  66 : 1048 : int32\_t sfw = params->getStartupMessage()->iSFSizeX;  67 : 1048 : uint32\_t numLayers = params->getStartupMessage()->uiSFLCountInDpb;  68 : 1048 : size\_t posCounter = 0;  69 [ + - ][ + + ]: 2152 : for( uint32\_t i = 0; i < numLayers; i ++ )  70 : : {  71 : 1104 : SFLayer::Pointer sfl = layers + i;  72 : 1104 : int64\_t sflX = sfl->getLLx();  73 : 1104 : uint32\_t numCols = sfl->getNoOfCols();  74 : 1104 : SFGroup::Pointer sfgs = sfl->getSFGroups();  75 [ + - + + ]: 9917 : for( uint32\_t j = 0; j < numCols; j++ )  76 : : {  77 : 8813 : int64\_t sfgX = sflX + j \* sfw;  78 : : WindowPosition::Pointer wp  79 : 8813 : = mPositions->getPositionForSFG( sfgX );  80 [ + + ]: 8813 : if( !wp   81 : : || (wp && ( sfgX - wp->getPosition() >= sfw )))  82 : : {  83 : 8548 : wp = new(std::nothrow) WindowPosition();  84 : : //#ifdef ERROR\_SEED //<<Test\_Id:Sliding\_Window\_wp\_Set\_To\_NULL>>  85 : : // wp = NULL;  86 : : //#endif  87 [ - + ]: 8548 : if( NULL == wp )  88 : : {  89 : 0 : int iErrNo = errno; //System error no  90 : 0 : char systemErrorMsg[BUFSIZ];  91 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  92 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  93 : : ErrorHandler::eErrMemoryAllocation,  94 : : pErr->getErrorMsg(  95 : : ErrorHandler::eErrMemoryAllocation,  96 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  97 : : }  98 : 8548 : mPositions->addPosition( wp );  99 : 8548 : wp->setPosition( sfgX );  100 : 8548 : wp->setIndex( posCounter );  101 : 8548 : posCounter++;  102 : :   103 : : }  104 : 8813 : wp->addTargetSFG( sfgs + j );  105 : : }  106 : : }  107 : : XDirection dir = (params->getStartupMessage()->usSFOrder <= 4)  108 : : ? eForwardXOrder  109 : 1048 : : eBackwardXOrder;  110 : 1048 : mPositions->setDirection( dir );  111 : 1048 : mPositions->sortPositions();  112 : 1048 : mCurrentStartIndex = 0;  113 : 1048 : mCurrentProcessedIndex = 0;  114 : 1048 : uint32\_t endIdx = params->getStartupMessage()->uiSlidingWindowSFGCount;  115 [ + + ]: 1048 : if( endIdx  116 : : && (params->getStartupMessage()->usFlag  117 : : & getFlagMask( FLAG\_MISC\_WINDOW\_AUTOCALC ) ) )  118 : : {  119 : 7 : endIdx = calculateWindowSizeInSFGs( params );  120 : : }  121 : :   122 [ - + ][ + + ]: 1048 : mCurrentEndIndex = ( endIdx == 0 ) ? posCounter - 1 : ((posCounter < endIdx) ? posCounter - 1: endIdx - 1);  [ # # ]  123 : : //If 0 comes from DPM, process everything  124 : : }  125 : :   126 : : template<typename \_Tp>  127 : : \_Tp mod( \_Tp inValue )  128 : 49 : {  129 [ + + ]: 49 : return ((inValue < 0 ) ? -1 : 1) \* inValue;  130 : : }  131 : :   132 : : uint32\_t  133 : : SlidingWindow::calculateWindowSizeInSFGs(  134 : 7 : const ConverterParams \*inParams) throw(DPL::Error \*) {  135 : : const DPMANAGER\_MESSAGE\_CONVERTER \*pStartMsg  136 : 7 : = inParams->getStartupMessage();  137 : 7 : const sgs\_column\_t \*pSecondMsg = inParams->getColumnParams();  138 : :   139 : 7 : int64\_t maxShift = 0;  140 : : //1. Read max from swing back params  141 [ + - ]: 7 : maxShift = (maxShift < mod( pStartMsg->iSwingBackOffsetX1))  142 : : ? mod( pStartMsg->iSwingBackOffsetX1) : maxShift;  143 [ + + ]: 7 : maxShift = (maxShift < mod( pStartMsg->iSwingBackOffsetX2))  144 : : ? mod( pStartMsg->iSwingBackOffsetX2) : maxShift;  145 [ + + ]: 7 : maxShift = (maxShift < mod( pStartMsg->iSwingBackOffsetX3))  146 : : ? mod( pStartMsg->iSwingBackOffsetX3) : maxShift;  147 [ + + ]: 7 : maxShift = (maxShift < mod( pStartMsg->iSwingBackOffsetX4))  148 : : ? mod( pStartMsg->iSwingBackOffsetX4) : maxShift;  149 [ + + ]: 7 : maxShift = (maxShift < mod( pStartMsg->iSwingBackOffsetX5))  150 : : ? mod( pStartMsg->iSwingBackOffsetX5) : maxShift;  151 : :   152 : : //2. Read size from GMC max correct  153 : 7 : maxShift += static\_cast<int64\_t>(pSecondMsg->gmc.max\_allowable\_value);  154 : :   155 : : //3. Consider user input  156 [ + + ]: 7 : maxShift = ( maxShift < pStartMsg->uiSlidingWindowSFGCount  157 : : \* pStartMsg->iSFSizeX )  158 : : ? pStartMsg->uiSlidingWindowSFGCount \* pStartMsg->iSFSizeX  159 : : : maxShift;  160 : : #ifdef INCLUDE\_CELL\_SIZE\_MAX  161 : : //4. Read max cell size  162 : : maxShift = (maxShift < pStartMsg->ullCellMaxSizeX)  163 : : ? pStartMsg->ullCellMaxSizeX : maxShift;  164 : : #endif  165 [ - + ][ # # ]: 7 : if( static\_cast<uint64\_t>(maxShift) > pStartMsg->ullCellMaxSizeX  166 : : && maxShift > pStartMsg->uiSlidingWindowSFGCount  167 : : \* pStartMsg->iSFSizeX )  168 : : {  169 : 0 : WarningHandler \*pWrn = WarningHandler::instance();  170 : 0 : pWrn->warningAdd( WarningHandler::eWrnShiftMoreThanNormal );  171 : : std::string msg = pWrn->getWarningMsg(  172 : : WarningHandler::eWrnShiftMoreThanNormal,  173 : : maxShift, pStartMsg->ullCellMaxSizeX,  174 : : pStartMsg->uiSlidingWindowSFGCount  175 : 0 : \* pStartMsg->iSFSizeX );   176 : 0 : Logger::instance()->log(LogMessageTable::eUTIL\_WARNING\_GENERAL\_MESSAGE,  177 : : msg.c\_str());  178 : : }  179 : 7 : int64\_t sfgs = maxShift / pStartMsg->iSFSizeX;  180 : 7 : sfgs += ( sfgs \* pStartMsg->iSFSizeX < maxShift ) ? 1 : 0;  181 : 7 : e9k\_debug("Auto calculated SFG count: %lld\n", sfgs );  182 : 7 : Logger::instance()->log(LogMessageTable::eWINDOW\_SFG\_COUNT, sfgs );  183 [ # # ]: 7 : return sfgs;  184 : : }  185 : :   186 : : void  187 : 8286 : SlidingWindow::processCurrentWindow() throw(DPL::Error \*) {  188 : :   189 : 8286 : HealthCheckCounter::instance()->incrCounter(  190 : : HealthCheckCounter::eUpdateFrequent |  191 : : HealthCheckCounter::eUpdateInfrequent |  192 : : HealthCheckCounter::eUpdateSeldom);  193 [ + + ]: 8286 : while( mCurrentProcessedIndex <= mCurrentEndIndex )  194 : : {  195 : : WindowPosition::Pointer pos  196 : 8478 : = mPositions->getPosition( mCurrentProcessedIndex );  197 : : //DELAYED\_MODE::DISTRIBUTE UNDISTRIBUTED PATTERNS  198 : 8478 : try  199 : : {  200 : 16956 : pos->applyForAllCells( boost::ref( mCellScheduler ) );  201 : : }  202 [ # # ]: 0 : catch( DPL::Error \*pChild )  203 : : {  204 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  205 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  206 : : ErrorHandler::eErrFunction,  207 : : pErr->getErrorMsg(  208 : : ErrorHandler::eErrFunction,  209 : : "WindowPosition::applyForAllCells"));  210 : : }  211 [ + + ]: 8478 : mCurrentProcessedIndex++;  212 : : }  213 [ # # ]: 8286 : return;  214 : : }  215 : :   216 : : void  217 : 8238 : SlidingWindow::move() throw(DPL::Error \*) {  218 : :   219 : 8238 : HealthCheckCounter::instance()->incrCounter(  220 : : HealthCheckCounter::eUpdateFrequent |  221 : : HealthCheckCounter::eUpdateInfrequent |  222 : : HealthCheckCounter::eUpdateSeldom);  223 : : WindowPosition::Pointer pos  224 : 8238 : = mPositions->getPosition( mCurrentStartIndex );  225 : 8238 : std::vector<SFGroup::Pointer> sfgs = pos->getTargetSFGs();  226 : 8238 : bool hasNonEmpty = false;  227 : 8238 : for( std::vector<SFGroup::Pointer>::iterator sfg  228 : 8238 : = sfgs.begin(); sfg != sfgs.end(); ++sfg )  229 : : {  230 : 8503 : if( !(\*sfg)->getIsEmpty() )  231 : : {  232 : 1316 : mOutputGenerator->addSFGToOutputQueue( \*sfg );  233 : 1316 : e9k\_debug("Added SFG %p to output\n", \*sfg);  234 : 1316 : hasNonEmpty = true;  235 : : }  236 : : }  237 [ + + ]: 8238 : if( hasNonEmpty )  238 : : {  239 : 1270 : e9k\_debug("WIN pos at %p has non-empty SFG\n", pos);  240 : : }  241 : 8238 : mCurrentStartIndex++;  242 [ + + ]: 8238 : if( mPositions->getNumPositions() > mCurrentEndIndex + 1 )  243 : : {  244 : 3803 : mCurrentEndIndex++;  245 : : }  246 : : }  247 : :   248 : : bool  249 [ + + ]: 9286 : SlidingWindow::hasUnprocessedPositions() {  250 : 0 : return mCurrentStartIndex  251 : : < mPositions->getNumPositions();  252 : : }  253 : :   254 : : void  255 : 1048 : SlidingWindow::cellDistribute() throw(DPL::Error \*) {  256 : :   257 : 1048 : HealthCheckCounter::instance()->incrCounter(  258 : : HealthCheckCounter::eUpdateFrequent |  259 : : HealthCheckCounter::eUpdateInfrequent);  260 : 1048 : ConverterParams \*pParams = ConverterParams::instance();  261 : 1048 : mDataProc = new VSBDataProcessor();  262 : 1048 : mDataProc->initialize( pParams->getCellCommonData(),  263 : : pParams->getCellData(), pParams->getRefData(),  264 : : pParams->getLinkData() );  265 : : const DPMANAGER\_MESSAGE\_CONVERTER \*pStartMsg  266 : 1048 : = pParams->getStartupMessage();  267 : 1048 : uint64\_t blkSizeX = pStartMsg->ullVchipBSizeX;  268 : 1048 : uint64\_t blkSizeY = pStartMsg->ullVchipBSizeY;  269 : 1048 : size\_t numBlockX = pStartMsg->ullDpbWidth / blkSizeX;  270 : 1048 : size\_t numBlockY = pStartMsg->ullDpbHeight / blkSizeY;  271 : : //TODO::Consider orientation while traversing blocks?  272 [ + - ][ + + ]: 2105 : for( size\_t i = 0; i < numBlockX; i++ )  273 : : {  274 [ + - ][ + + ]: 2138 : for( size\_t j = 0; j < numBlockY; j++ )  275 : : {  276 : 1081 : mDataProc->setBlockIDs( i, j );  277 : 2291 : while(true)  278 : : {  279 : 2291 : Cell \*pCell = new Cell();  280 : 2291 : bool cellfound = false;  281 : 2291 : try  282 : : {  283 : 2291 : cellfound = mDataProc->getNextCell(pCell);  284 : : }  285 [ # # ]: 0 : catch (DPL::Error \*pChild)  286 : : {  287 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  288 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  289 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  290 : : pErr->getErrorMsg(   291 : : ErrorHandler::eErrFunction,  292 : : "SlidingWindow::cellDistribute"));  293 : : }  294 : :   295 [ + + ]: 2291 : if( cellfound )  296 : : {  297 : 1210 : if( pCell->isArray() )  298 : : {  299 : 47 : SDDB \*pSDDB = SDDB::instance();  300 : 47 : try   301 : : {  302 : 47 : pSDDB->ArrayCellFound(\*pCell);  303 : : }  304 [ # # ]: 0 : catch(DPL::Error \*pChild)  305 : : {  306 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  307 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  308 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  309 : : pErr->getErrorMsg(   310 : : ErrorHandler::eErrFunction,  311 : : "SDDB::ArrayCellFound"));  312 : : }  313 : :   314 : : //Array Cell flattening  315 : 47 : IntPosition inBlockPos(i, j);  316 : 47 : IntPosition outBlockPos;  317 : :   318 : 47 : FlattenedContext flattenContext;  319 : 47 : flattenContext.setIsMoreCell(true);  320 : 47 : flattenContext.setNumCellX(0);  321 : 47 : flattenContext.setNumCellY(0);  322 : 187 : while(true)  323 : : {  324 : 187 : Cell \*outCell = new Cell();   325 [ + + ]: 187 : if(!mDataProc->getNextFlattenedCell(  326 : : \*pCell, outCell, inBlockPos, outBlockPos,  327 : : &flattenContext))  328 : : {  329 [ + - ]: 47 : delete outCell;  330 : 140 : break;  331 : : }  332 : 140 : try  333 : : {  334 : 140 : scheduleSingleCell( outCell );  335 : : }  336 [ # # ]: 0 : catch(DPL::Error \*pChild)  337 : : {  338 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  339 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  340 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  341 : : pErr->getErrorMsg(   342 : : ErrorHandler::eErrFunction,  343 : : "SlidingWindow::scheduleSingleCell"));  344 : : }  345 : 140 : try  346 : : {  347 : 140 : pSDDB->SingleCellExtractedFromArrayCell(outCell);  348 : : }  349 [ # # ]: 0 : catch(DPL::Error \*pChild)  350 : : {  351 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  352 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  353 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  354 : : pErr->getErrorMsg(   355 : : ErrorHandler::eErrFunction,  356 : : "SDDB::SingleCellExtractedFromArrayCell"));  357 : : }  358 : :   359 : : }  360 : 47 : try  361 : : {  362 : 47 : pSDDB->ArrayCellFlatteningCompleted();  363 : : }  364 [ # # ]: 0 : catch(DPL::Error \*pChild)  365 : : {  366 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  367 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  368 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  369 : : pErr->getErrorMsg(  370 : : ErrorHandler::eErrFunction,  371 : : "SDDB::ArrayCellFlatteningCompleted"));  372 : : }  373 : :   374 [ + - ]: 47 : delete pCell;  375 : : }  376 : : else //Cell is single cell  377 : : {  378 : 1163 : try  379 : : {  380 : 1163 : scheduleSingleCell( pCell );  381 : : }  382 [ # # ]: 0 : catch(DPL::Error \*pChild)  383 : : {  384 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  385 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  386 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  387 : : pErr->getErrorMsg(   388 : : ErrorHandler::eErrFunction,  389 : : "SlidingWindow::scheduleSingleCell"));  390 : : }  391 : : }  392 : : }  393 : : else  394 : : {  395 [ + - ]: 1081 : delete pCell;  396 : 1081 : break;  397 : : }  398 : : }  399 : : }  400 : : }  401 : : //Cell scheduling complete  402 [ # # ]: 1048 : nftFree(mThreadDataSizes);  403 : : }  404 : :   405 : : void  406 : 1303 : SlidingWindow::scheduleSingleCell( Cell \*inCell ) throw(DPL::Error \*) {  407 : :   408 : 1303 : HealthCheckCounter::instance()->incrCounter(  409 : : HealthCheckCounter::eUpdateFrequent);  410 : 1303 : ConverterParams \*pParams = ConverterParams::instance();  411 : : const DPMANAGER\_MESSAGE\_CONVERTER \*pStartMsg  412 : 1303 : = pParams->getStartupMessage();  413 : 1303 : if( !mBBox.checkIsOverlapped(  414 : : LongPosition(  415 : : inCell->startPos.getX() + pStartMsg->llDpbX,  416 : : inCell->startPos.getY() + pStartMsg->llDpbY),  417 : : UIntPosition( inCell->sc.dx, inCell->sc.dy ) ) )  418 : : {  419 : 59 : Logger::instance()->log(LogMessageTable::eSHOT\_DISCARD,  420 : : inCell->startPos.getX(), inCell->startPos.getY());  421 : 59 : e9k\_debug("Discarded cell at: %d %d\n",  422 : : inCell->startPos.getX(), inCell->startPos.getY() );  423 [ + - ]: 59 : delete inCell;  424 : 1244 : return;  425 : : }  426 : : int64\_t adjustment = ( eForwardXOrder == mPositions->getDirection() )  427 : 1244 : ? 0 : inCell->sc.dx;  428 : : WindowPosition::Pointer pos  429 : : = mPositions->getWindowPositionForX(  430 : 1244 : inCell->startPos.getX() + adjustment );  431 [ - + ]: 1244 : if( !pos )  432 : : {  433 [ # # ]: 0 : delete inCell;  434 : 1244 : return;  435 : : }  436 : : //1. Assign proper thread ID to cell  437 : 1244 : size\_t dataSize = inCell->getSegmentDataSize();  438 : : uint16\_t numThreads = ConverterParams::instance()->getStartupMessage()  439 : 1244 : ->usConcurrency;  440 [ + + ]: 1244 : if( 0 == mThreadDataSizes )  441 : : {  442 : 1038 : try  443 : : {  444 : 1038 : mThreadDataSizes = (size\_t \*) nftMalloc( sizeof(size\_t)  445 : : \* numThreads  446 : : \* mPositions->getNumPositions());  447 : : }  448 [ # # ]: 0 : catch(DPL::Error \*pChild)  449 : : {  450 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  451 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  452 : : ErrorHandler::eErrFunction,  453 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  454 : : "nftMalloc"));  455 : : }  456 : 1244 : bzero( mThreadDataSizes, sizeof(size\_t) \* numThreads  457 : : \* mPositions->getNumPositions());  458 : : }  459 : 1244 : size\_t minDataId = 0;  460 : 1244 : size\_t minDataSize = std::numeric\_limits<size\_t>::max();  461 : 1244 : size\_t posIdx = pos->getIndex();  462 [ + - ][ + + ]: 2536 : for( size\_t i = 0; i < numThreads; i ++ )  463 : : {  464 : 1292 : size\_t \*pDataSize = mThreadDataSizes + posIdx \* numThreads + i;  465 [ + + ]: 1292 : if( \*pDataSize + dataSize < minDataSize )  466 : : {  467 : 1244 : minDataSize = \*pDataSize + dataSize;  468 : 1244 : minDataId = i;  469 : : }  470 : : }  471 : 1244 : size\_t \*pDataSize = mThreadDataSizes + posIdx \* numThreads + minDataId;  472 : 1244 : \*pDataSize = minDataSize;  473 : 1244 : inCell->setTargetThreadID( minDataId );  474 : 1244 : e9k\_debug("Cell will be scheduled to thread %llu\n", minDataId );  475 : :   476 : : //2. Assign cell to proper window position  477 : 1244 : try  478 : : {  479 : 1244 : pos->addCell( inCell );  480 : 1244 : e9k\_debug("Cell at %lld added to Window Pos at %lld (%p)\n",  481 : : inCell->startPos.getX(), pos->getPosition(), pos);  482 : : }  483 [ # # ]: 0 : catch(DPL::Error \*pChild)  484 : : {  485 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  486 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  487 : : ErrorHandler::eErrFunction,  488 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  489 : : "WindowPosition::addCell"));  490 : : }  491 : : }  492 : :   493 : :   494 : : void  495 : : SlidingWindow::addCellProcessor(  496 : 1096 : FixedModeCellProcessor::Pointer inProcessor) {  497 : 1096 : mCellProcessors.push\_back( std::make\_pair(inProcessor,false) );  498 : : }  499 : :   500 : : void  501 : 1048 : SlidingWindow::setOutputGenerator(OutputGenerator::Pointer value) {  502 : 1048 : mOutputGenerator = value;  503 : : }  504 : :   505 : : void  506 : 1048 : SlidingWindow::operator()() throw() {  507 : : {  508 : 1048 : ConverterParams \*pParams = ConverterParams::instance();  509 : : const DPMANAGER\_MESSAGE\_CONVERTER \*pStartMsg  510 : 1048 : = pParams->getStartupMessage();  511 : : LongPosition bbLLCorner( pStartMsg->llPtnProcessOriginX,  512 : 1048 : pStartMsg->llPtnProcessOriginY );  513 : : ULongPosition bbSize( pStartMsg->ullPtnProcessWidth,  514 : 1048 : pStartMsg->ullPtnProcessHeight );  515 : 1048 : mBBox.setLowerLeftCorner( bbLLCorner );  516 : 1048 : mBBox.setSize( bbSize );  517 : : }  518 : 1048 : try  519 : : {  520 : 1048 : generateWindowPositions();  521 : 0 : e9k\_debug("Window positions generated\n");  522 : : }  523 [ # # ]: 0 : catch( DPL::Error \*pChild )  524 : : {  525 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  526 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  527 : : ErrorHandler::eErrFunction,  528 : : pErr->getErrorMsg(  529 : : ErrorHandler::eErrFunction, "generateWindowPositions"),  530 : : getThreadManager());  531 : 1048 : return;  532 : : }  533 : :   534 : 1048 : try  535 : : {  536 : 1048 : cellDistribute();  537 : : }  538 [ # # ]: 0 : catch( DPL::Error \*pChild )  539 : : {  540 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  541 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  542 : : ErrorHandler::eErrFunction,  543 : : pErr->getErrorMsg(  544 : : ErrorHandler::eErrFunction,  545 : : "SlidingWindow::cellDistribute"),  546 : : getThreadManager());  547 : 0 : return;  548 : : }  549 : :   550 : 8286 : while( hasUnprocessedPositions() )  551 : : {  552 : 8286 : HealthCheckCounter::instance()->incrCounter(  553 : : HealthCheckCounter::eUpdateFrequent |  554 : : HealthCheckCounter::eUpdateInfrequent);  555 : :   556 : 8286 : try  557 : : {  558 : 8286 : processCurrentWindow();  559 : : }  560 [ # # ]: 0 : catch( DPL::Error \*pChild )  561 : : {  562 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  563 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  564 : : ErrorHandler::eErrFunction,  565 : : pErr->getErrorMsg(  566 : : ErrorHandler::eErrFunction,  567 : : "SlidingWindow::process"),  568 : : getThreadManager());  569 : 0 : return;  570 : : }  571 : 8286 : int i = 0;  572 : : {  573 : 8286 : boost::mutex::scoped\_lock l(mMutex);  574 : 8286 : for( FixedModeCellProcessorList::iterator it  575 : 8286 : = mCellProcessors.begin(); it != mCellProcessors.end();  576 : : it++ )  577 : : {  578 : 8526 : SchedulingInfo cellProcInfo = \*it;  579 [ + + ]: 8526 : if( cellProcInfo.second )  580 : : {  581 : : FixedModeCellProcessor::Pointer cellProc  582 : 1066 : = cellProcInfo.first;  583 : 1066 : cellProc->notifyCellSchedulingCompletion();  584 : 1066 : mNumActiveProcessors++;  585 : 1066 : cellProcInfo.second = false;  586 : 1066 : mCellProcessors[i] = cellProcInfo;  587 : 1066 : e9k\_debug("Notifying processor of cell availability\n");  588 : : }  589 : 8526 : i++;  590 : : }  591 [ + - ][ + + ]: 8286 : while( !mStop && 0 != mNumActiveProcessors )  592 : : {  593 : 1066 : mWaitCondition.wait( l );  594 : : }  595 : 48 : if( mStop )  596 : : {  597 : : //Somebody requested stop  598 : 48 : return;   599 : : }  600 : : }  601 : 8238 : move();  602 : : }  603 : 1000 : mOutputGenerator->setIsComplete( true );  604 [ # # ]: 1048 : return;  605 : : }  606 : :   607 : : void  608 : 1018 : SlidingWindow::notifyProcessingCompletion() {  609 : 1018 : boost::mutex::scoped\_lock l( mMutex );  610 : 1018 : mNumActiveProcessors--;  611 : 1018 : mWaitCondition.notify\_one();  612 : : }  613 : :   614 : : void  615 : 1048 : SlidingWindow::stop() {  616 : 1048 : boost::mutex::scoped\_lock l( mMutex );  617 : 1048 : mStop = true;  618 : 1048 : mWaitCondition.notify\_one();  619 : : }  620 : :   621 : :   622 : 1215 : } // namespace nft |

|  |
| --- |
| Generated by: [LCOV version 1.9](http://ltp.sourceforge.net/coverage/lcov.php) |