| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [Common/include](http://docs.google.com/index.html) - BufferType.h (source / [functions](http://docs.google.com/BufferType.h.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 44 | 61 | 72.1 % | | Date: | 2012-03-23 |  | Functions: | 4 | 6 | 66.7 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 20 | 36 | 55.6 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #ifndef NFT\_BUFFERTYPE\_H  2 : : #define NFT\_BUFFERTYPE\_H  3 : :   4 : :   5 : : #include <stdint.h>  6 : : #include <cstdlib>  7 : : #include <memory>  8 : :   9 : : #include "DPL.h"  10 : :   11 : : #include "ErrorHandler.h"  12 : : #include "Memory.h"  13 : : #include "MemMgr.h"  14 : : #include "ThreadID.h"  15 : : #include "Logger.h"  16 : :   17 : : namespace nft {  18 : :   19 : : template<typename \_ShotType>  20 : : class BufferType {  21 : : public:  22 : : typedef \_ShotType PatternType;  23 : :   24 : : private:  25 : : struct Node : public NftNewDeleteBase  26 : : {  27 : : const size\_t cSizeIncrement;  28 : : PatternType \*mPatterns;  29 : : uint32\_t mShotCount;  30 : : uint32\_t mMaxShotCount;  31 : :   32 : : Node()  33 : 10471 : :cSizeIncrement(2),  34 : : mPatterns(NULL),  35 : : mShotCount(0),  36 : : mMaxShotCount(0)  37 : 20942 : {  38 : : }  39 : : };  40 : :   41 : : public:  42 : : static size\_t MemSize;  43 : :   44 : : PatternType \*  45 : : getMemoryForShots(ThreadID inThreadID,  46 : : uint32\_t inNumShots = 1) throw(DPL::Error \*);  47 : :   48 : : inline const PatternType \*  49 : : getPatterns(ThreadID inThreadID) const;  50 : :   51 : : inline uint64\_t  52 : : getShotCount(ThreadID inThreadID) const;  53 : :   54 : : void  55 : : releaseMemory();  56 : :   57 : :   58 : : BufferType( ThreadID inNumThreads);  59 : :   60 : :   61 : : ~BufferType();  62 : :   63 : : BufferType(const BufferType<\_ShotType> & source);  64 : :   65 : : BufferType<\_ShotType> &  66 : : operator=(const BufferType<\_ShotType> & source);  67 : :   68 : : private:  69 : : uint16\_t mNumThreads;  70 : : //A 2-D pattern buffer. Each element in the 1st dimension contains a list of shots.  71 : : //The shots in the list are added to by a particular thread.  72 : : Node \* mPatterns;  73 : : };  74 : :   75 : : template<typename \_ShotType>  76 : : typename BufferType<\_ShotType>::PatternType \*  77 : : BufferType<\_ShotType>::getMemoryForShots(  78 : 15956 : ThreadID inThreadID, uint32\_t inNumShots) throw(DPL::Error \*) {  79 : : //#ifdef ERROR\_SEED //<<Test\_Id:mPatterns\_Set\_To\_NULL>>  80 : : // mPatterns=NULL;  81 : : //#endif  82 [ - + ]: 15956 : if(NULL == mPatterns)  83 : : {  84 : 0 : int iErrNo = errno; //System error no  85 : 0 : char systemErrorMsg[BUFSIZ];  86 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  87 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  88 : : ErrorHandler::eErrMemoryAllocation,  89 : : pErr->getErrorMsg(  90 : : ErrorHandler::eErrMemoryAllocation,  91 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  92 : : }  93 : :   94 : : // invalid thread ID  95 [ - + ]: 15956 : if(inThreadID >= mNumThreads)  96 : : {  97 : 0 : return NULL;  98 : : }  99 : 15956 : uint32\_t prevShotCnt = mPatterns[inThreadID].mShotCount;  100 : :   101 [ + + ]: 15956 : if(mPatterns[inThreadID].mShotCount < mPatterns[inThreadID].mMaxShotCount)  102 : : {  103 : : // memory already allocated  104 [ + - ]: 3588 : if(NULL != mPatterns[inThreadID].mPatterns)  105 : : {  106 : 3588 : mPatterns[inThreadID].mShotCount += inNumShots;  107 : 3588 : return reinterpret\_cast<PatternType \*>  108 : : (mPatterns[inThreadID].mPatterns + prevShotCnt);  109 : : }  110 : : else  111 : : {  112 : : // handled later  113 : : }  114 : : }  115 : 12368 : MemSize -= mPatterns[inThreadID].mMaxShotCount \* sizeof(PatternType);  116 : 12368 : do  117 : : {  118 [ - + ]: 12368 : mPatterns[inThreadID].mMaxShotCount  119 : : += mPatterns[inThreadID].cSizeIncrement;  120 : : }  121 : : while( prevShotCnt + inNumShots > mPatterns[inThreadID].mMaxShotCount);  122 : :   123 : 12368 : PatternType \*tempLoc = NULL;  124 : 12368 : try  125 : : {  126 : 12368 : size\_t oldSize = prevShotCnt \* sizeof(PatternType);  127 : 12368 : tempLoc = (PatternType \*)MemMgr::instance()->getMemory(  128 : : mPatterns[inThreadID].mMaxShotCount  129 : : \* sizeof(PatternType));  130 : 12368 : ::memmove( tempLoc, mPatterns[inThreadID].mPatterns,oldSize);  131 [ + + ]: 12368 : if(prevShotCnt){  132 : 1900 : MemMgr::instance()->releaseMemory(  133 : : mPatterns[inThreadID].mPatterns,oldSize);  134 : : }  135 : :   136 : 12368 : MemSize += mPatterns[inThreadID].mMaxShotCount \* sizeof(PatternType);  137 : : }  138 [ # # ]: 0 : catch( DPL::Error \*pChild )  139 : : {  140 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  141 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  142 : : ErrorHandler::eErrFunction,  143 : : pErr->getErrorMsg(  144 : : ErrorHandler::eErrFunction, "nftRealloc"));  145 : : }  146 : :   147 : 12368 : mPatterns[inThreadID].mPatterns = tempLoc;  148 : 12368 : mPatterns[inThreadID].mShotCount += inNumShots;  149 [ # # ]: 15956 : return reinterpret\_cast<PatternType \*>(  150 : : mPatterns[inThreadID].mPatterns + prevShotCnt);  151 : : }  152 : :   153 : : template<typename \_ShotType>  154 : : inline const typename BufferType<\_ShotType>::PatternType \*  155 : 21172 : BufferType<\_ShotType>::getPatterns(ThreadID inThreadID) const {  156 [ - + ]: 10586 : if(inThreadID >= mNumThreads)  157 : 0 : {  158 : : return NULL;  159 : : }  160 : : else  161 : 10586 : {  162 : : return mPatterns[inThreadID].mPatterns;  163 : : }  164 : : }  165 : :   166 : : template<typename \_ShotType>  167 : : inline uint64\_t  168 : 21172 : BufferType<\_ShotType>::getShotCount(ThreadID inThreadID) const {  169 [ - + ]: 10586 : if(inThreadID >= mNumThreads)  170 : 0 : {  171 : : return 0;  172 : : }  173 : : else  174 : 10586 : {  175 : : return mPatterns[inThreadID].mShotCount;  176 : : }  177 : : }  178 : :   179 : : template<typename \_ShotType>  180 : : BufferType<\_ShotType>::BufferType(ThreadID inNumThreads)  181 : 10423 : : mNumThreads(inNumThreads),  182 : 10423 : mPatterns(NULL) {  183 [ + - ][ + + ]: 10471 : mPatterns = new (std::nothrow) Node[mNumThreads];  184 : : }  185 : :   186 : : template<typename \_ShotType>  187 : 20846 : BufferType<\_ShotType>::~BufferType() {  188 : : // release memory allocated by threads   189 [ + - ]: 10423 : delete [] mPatterns;  190 : 10423 : mPatterns = NULL;  191 : : }  192 : :   193 : : template<typename \_ShotType>  194 : : BufferType<\_ShotType>::BufferType(const BufferType<\_ShotType> & source)  195 : 0 : :mNumThreads( source.mNumThreads ),  196 : 0 : mPatterns( source.mPatterns ) {  197 : : }  198 : :   199 : : template<typename \_ShotType>  200 : : BufferType<\_ShotType> &  201 : 0 : BufferType<\_ShotType>::operator=(const BufferType<\_ShotType> & source) {  202 [ # # ]: 0 : if(this != &source)  203 : : {  204 : 0 : mNumThreads = source.mNumThreads;  205 : 0 : mPatterns = source.mPatterns;  206 : : }  207 : 0 : return \*this;  208 : : }  209 : :   210 : : template<typename \_ShotType>  211 : 10535 : void BufferType<\_ShotType>::releaseMemory() {  212 [ + - ]: 10535 : if(mPatterns != NULL)  213 : : {  214 : : // loop over threads  215 : : // release memry allocated for nodes handled by each thread  216 [ + - ][ + + ]: 21118 : for(int i = 0; i < mNumThreads; i++)  217 : : {  218 [ + + ]: 10583 : if(mPatterns[i].mPatterns != NULL)  219 : : {  220 : : #if 0   221 : : nftFree(mPatterns[i].mPatterns);  222 : : #endif   223 : : //#if 0 //PERF (Memory manager)   224 : 10468 : MemMgr::instance()->releaseMemory(  225 : : mPatterns[i].mPatterns,  226 : : mPatterns[i].mMaxShotCount  227 : : \* sizeof( PatternType ) );  228 : : //#endif   229 : 10468 : mPatterns[i].mPatterns = NULL;  230 : 10468 : mPatterns[i].mMaxShotCount = 0;  231 : 10468 : mPatterns[i].mShotCount = 0;  232 : : }  233 : : }  234 : : }  235 : : }  236 : :   237 : : } // namespace nft  238 : : #endif |

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