```
#ifndef SMPCACHE_H
#define SMPCACHE_H

#include "libcore/MemObj.h"
#include "SMemorySystem.h"
#include "SMPProtocol.h"
#include "SMPMemRequest.h"
#include "SMPCacheState.h"
#include "SMPSystemBus.h"
#include "MSHR.h"
#include "Port.h"

#ifdef SESC_ENERGY
#include "GEnergy.h"
#endif

#include "vector"
#include "estl.h"
#include <map>

class SMPCache : public MemObj {
public:
    typedef CacheGeneric<SMPCacheState, PAddr, false>
CacheType;
    typedef CacheGeneric<SMPCacheState, PAddr, false>::CacheLine
Line;
```

```
private:
        static const char *cohOutfile;

    void processReply(MemRequest *mreq);
    //void resolveSituation(SMPMemRequest *sreq);
protected:


    CacheType *cache;

    PortGeneric *cachePort;
    TimeDelta_t missDelay;
    TimeDelta_t hitDelay;

    MSHR<PAddr, SMPCache> *outsReq; // buffer for requests coming
from upper levels
    //static MSHR<PAddr, SMPCache> *mutExclBuffer;


    class Entry {
    public:
        int32_t outsResps;        // outstanding responses: number
of caches
        // that still need to acknowledge invalidates
        bool invalidate;
        bool writeback;
        CallbackBase *cb;
        Entry() {
            outsResps = 0;
            cb = 0;
            invalidate = false;
            writeback = false;
        }
    };

    typedef HASH_MAP<PAddr, Entry> PendInvTable;

    PendInvTable pendInvTable; // pending invalidate table

    // BEGIN statistics
    GStatsCntr readHit;
    GStatsCntr writeHit;
    GStatsCntr readMiss;
    GStatsCntr writeMiss;
    GStatsCntr readHalfMiss;  // attention: these half misses have
a != semantic
    GStatsCntr writeHalfMiss; // than Cache.cpp: these counts are
included in
    // other counters because MSHR is used differently
    GStatsCntr writeBack;
    GStatsCntr linePush;
    GStatsCntr lineFill;
    GStatsCntr readRetry;
```

```
    GStatsCntr writeRetry;

    GStatsCntr compMiss;
    GStatsCntr confMiss;
    GStatsCntr capMiss;

    std::unordered_set<PAddr> accessedBlocks;
    std::list<PAddr> lruCache;

    GStatsCntr invalDirty;
    GStatsCntr allocDirty;

#ifdef SESC_ENERGY
    static unsigned cacheID;
    unsigned myID;
    GStatsEnergy *rdEnergy[2]; // 0 hit, 1 miss
    GStatsEnergy *wrEnergy[2]; // 0 hit, 1 miss
#endif

    // END statistics

    SMPProtocol *protocol;

    // interface with upper level
    void read(MemRequest *mreq);
    void write(MemRequest *mreq);
    void pushline(MemRequest *mreq);
    void specialOp(MemRequest *mreq);

    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::read> readCB;
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::write> writeCB;
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::specialOp> specialOpCB;

    // port usage accounting
    Time_t nextSlot() {
        return cachePort->nextSlot();
    }

    // local routines
    void doRead(MemRequest *mreq);
    // JJO
    void doReadRemote(MemRequest *mreq);
    void doWrite(MemRequest *mreq);
    void doPushLine(MemRequest *mreq);
    void doWriteBack(PAddr addr);
    void concludeWriteBack(Time_t initialTime);
    void sendRead(MemRequest* mreq);
    void sendWrite(MemRequest* mreq);

    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::doRead> doReadCB;
```

```
    // JJO
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::doReadRemote> doReadRemoteCB;
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::doWrite> doWriteCB;
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::doPushLine> doPushLineCB;
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::sendRead> sendReadCB;
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::sendWrite> sendWriteCB;
    typedef CallbackMember1<SMPCache, Time_t,
            &SMPCache::concludeWriteBack> concludeWriteBackCB;

public:

    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::sendRead> doReadAgainCB;
    typedef CallbackMember1<SMPCache, MemRequest *,
            &SMPCache::sendWrite> doWriteAgainCB;

    SMPCache(SMemorySystem *gms, const char *section, const char
*name);
    ~SMPCache();

        static void PrintStat();

#if (defined SIGDEBUG)
    void pStat();

#endif

    //static std::set<SMPMemRequest*> detourSet;
    //static std::map<SMPMemRequest*, SMPMemRequest*> replaceMap;

    // JJO
    //static bool msgPrinted;
    void doWriteAgain(MemRequest *mreq);

    //static HASH_MAP<PAddr, std::list<CallbackBase*> > pendingList;

    //static std::map<PAddr, MemRequest* > mutInvReq;

    static unsigned int dlcnt;

    int32_t maxNodeID;
        int32_t maxNodeID_bit;
        int32_t nodeSelSht;
    inline int32_t getMaxNodeID() { return maxNodeID; }
        inline int32_t getMaxNodeID_bit() { return maxNodeID_bit; }
    inline int32_t getNodeID() { return nodeID; }
    int32_t getHomeNodeID(PAddr addr);
    int32_t getL2NodeID(PAddr addr);
```

```
    //std::map<PAddr, bool> pendingWriteBackReq;
    std::map<PAddr, int32_t> pendingInvCounter;
    std::map<PAddr, int32_t> invCounter;
    //std::map<PAddr, bool> pendingReplyFlag;
    std::map<PAddr, bool> writeBackPending;
    std::set<PAddr> pendingInv;
        std::map<PAddr, bool> replyReady;
        std::map<PAddr, Time_t> replyReadyTime;
    HASH_MAP<PAddr, CallbackBase* > pendRemoteRead;


    //void updateDirectory(SMPMemRequest *sreq);
    //void sendUpdateDirectory(SMPMemRequest *sreq);
    //typedef CallbackMember1<SMPCache, SMPMemRequest *,
    //                        &SMPCache::updateDirectory>
doUpdateDirectoryCB;

    // BEGIN MemObj interface

    bool isCache() const {
        return true;
    }
    //const bool isCache() const { return true; }

    // port availability
    Time_t getNextFreeCycle() const;

    // interface with upper level
    bool canAcceptStore(PAddr addr);
    void access(MemRequest *mreq);
    // JJO
    void remoteAccess(MemRequest *mreq);
    void sendInvDirUpdate(PAddr addr, PAddr new_addr, CallbackBase
*cb, bool wb, bool data);
    void processInvDirAck(SMPMemRequest *sreq);

    // interface with lower level
    void returnAccess(MemRequest *mreq);

    void invalidate(PAddr addr, ushort size, MemObj *oc);
    void doInvalidate(PAddr addr, ushort size);
    void realInvalidate(PAddr addr, ushort size, bool writeBack);

    // END MemObj interface

    // BEGIN protocol interface

    // interface used by protocol to access lower level
    void sendBelow(SMPMemRequest *sreq);
    void sendBelowI(SMPMemRequest *sreq);
    void respondBelow(SMPMemRequest *sreq);
    void receiveFromBelow(SMPMemRequest *sreq);
    void doReceiveFromBelow(SMPMemRequest *sreq);
```

```
    typedef CallbackMember1<SMPCache, SMPMemRequest *,
            &SMPCache::doReceiveFromBelow> doReceiveFromBelowCB;

    // interface used by protocol to access upper level
    void concludeAccess(MemRequest *mreq);

    // interface used by protocol to operate on cache lines
    Line *getLine(PAddr addr);
    void writeLine(PAddr addr);
    void invalidateLine(PAddr addr, CallbackBase *cb, bool writeBack
= false);
    Line *allocateLine(PAddr addr, CallbackBase *cb, bool
canDestroyCB = true);
    void doAllocateLine(PAddr addr, PAddr rpl_addr, CallbackBase
*cb);

    typedef CallbackMember3<SMPCache, PAddr, PAddr, CallbackBase *,
            &SMPCache::doAllocateLine> doAllocateLineCB;

    PAddr calcTag(PAddr addr) {
        return cache->calcTag(addr);
    }

    // END protocol interface

    // debug function
#ifdef SESC_SMP_DEBUG
    Line* findLine(PAddr addr) {
        return cache->findLine(addr);
    }
    void inclusionCheck(PAddr addr);
#endif
};


#endif // SMPCACHE_H
```