Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Simulace hierarchie sdílených pamětí cache

Bc. Jindřich Čapek

Vedoucí práce: Ing. Jiří Kašpar

29. dubna 2015

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 29. dubna 2015 .....

Odkaz na tuto práci

Čapek, Jindřich. *Simulace hierarchie sdílených pamětí cache.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce obsahuje implementaci hierarchie sdílených pamětí cache s koherenčním protokolem MOESI v simulátoru GEM5. Práce obsahuje krátký popis simulátoru GEM5 a jeho paměťového systému, popis koherenčního protokolu MOESI, popis implementace a propojení jednotlivých cache pamětí. Výstupem této práce je simulace systému, který obsahuje dva osmijádrové procesory s tříúrovňovou hierarchií pamětí cache. Ověření správnosti implementace jsme provedli pomocí testů obsažených v simulátoru GEM5. Dále jsme simulovali běh vybraných benchmarků ze sad benchmarků PARSEC a SPLASH-2 a výsledky porovnali s během na reálném systému.

Klíčová slova MOESI protokol, cache koherence, cache hierarchie, GEM5

Abstract

This thesis contains implementation of Shared Cache Hierarchy with cache coherency protocol MOESI in GEM5 simulator. This thesis contains short description of GEM5 simulator and its memory system, description of MOESI coherency protocol, description of implementation and connection between cache controllers. We simulated a system with two eight-core processors with three-level cache hierarchy. Implementation was tested with test embedded in GEM5 simulator. We ran selected benchmarks from PARSEC and SPLASH-2 suits and compared results with real system.

Keywords MOESI protocol, cache coherency, cache hierarchy, GEM5

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cílem této diplomové práce je implementovat hierarchii sdílených pamětí cache do simulátoru GEM5. Pro udržení koherence sdílených pamětí cache je implementován koherenční protokol MOESI s integrovanou adresářovou strukturou. První kapitola obsahuje popis simulátoru GEM5 se zaměřením na části potřebné pro implementaci koherenčního protokolu. V druhé kapitole je popsán MOESI protokol. Implementace je popsána ve třetí kapitole. V poslední kapitole je popsáno testování pomocí nástrojů v simulátoru GEM5 a pomocí sad benchmarků PARSEC a SPLASH-2. V příloze jsou uvedeny přechodové tabulky řadičů pamětí cache.

Koherenční protokoly

Koherenční protokoly zajišťují správný přístup ke sdílené paměti. Zajišťují, aby do každého datového bloku mohlo zapisovat pouze jedno jádro nebo aby datový blok mohlo číst více jader, ale nikoliv do něj zapisovat. Koherenční protokoly se dělí do dvou skupin. První skupinou jsou „update“ protokoly, které jsou založené na write-through policy, rozesílají všechny zápisy všem ostatním jádrům. Jejich nevýhodou je velmi špatná škálovatelnost a v moderních architekturách se nepoužívají. Druhou skupinou jsou invalidační protokoly, které se dále dělí na:

- „Snooping“ protokoly – neuchovávají informace o sdílení, komunikace probíhá mezi všemi paměťmi cache
- Protokoly založené na adresáři – informace o sdílení se uchovává v adresáři, komunikace probíhá jen s uzly, které jsou v uvedeny v adresáři. Procesor navíc obsahuje řadič adresáře, který registruje rozmístění datových bloků z paměti připojené k čipu procesoru. Řadič tedy ví s kým musí komunikovat a nemusí zdržovat cache, které daný datový blok neobsahují. Toto řešení škáluje pro větší počet procesorů podstatně lépe

než snooping protokoly. V této práci je implementován protokol založený na adresáři.

Mezi nejznámější protokoly patří: MESI, MESIF a MOESI. MESI protokol byl používan v procesorech Intel do uvedení MESIF protokolu, který má navíc stav „Forward“. Tento stav je speciální případ sdíleného stavu a označuje danou cache jako odesílatele odpovědi na požadavek.

MOESI protokol se používá v procesorech AMD. Tento protokol má navíc stav „Owned“ (vlastník) a umožňuje přechod z modifikovaného stavu do sdíleného bez zápisu modifikovaných dat do paměti.

Cíle

Cíle teto práce jsou:

- Implementace hierarchicky sdílené paměti cache s koherenčním protokolem MOESI do simulátoru GEM5.
- Funkční simulace dvou osmijádrových procesorů.
- Ověření implementace pomocí testů obsažených v simulátoru GEM5.
- Ověření a porovnání výkonu s reálným systémem pomocí sad benchmarků PARSEC a SPLASH-2.

Kapitola 1

Simulátor GEM5

Simulátor GEM5 [1] je společný projekt vytvořený společnostmi ARM, AMD, Intel aj. Vznikl spojením projektu M5 system simulator (simulátor procesoru) a GEMS memory system simulator (simulátor paměťového systému).

Simulátor podporuje dva režimy simulace. V režimu *full system simulation* je možné spustit nemodifikovanou linuxovou distribuci a v ní spustit libovolnou aplikaci. Tento režim je značně pomalý. Druhý režim *syscall emulation* umožňuje spustit staticky slinkovaný program, kde systémová volání jsou emulována. Hlavní výhodou této metody je rychlost, protože není potřeba startovat operační systém. Tato metoda má hlavní nevýhodu v tom, že není dostupná implementace velkého množství systémových volání a tím je nemožné simulovat běh některých aplikací.

1.1 Modely CPU

Simulátor GEM5 obsahuje tři modely procesoru:

- SimpleCPU – simulace jednoduchého procesoru bez pipeline
- InOrder – simulace procesoru s pipeline, provádění instrukcí „in order“ (instrukce jsou prováděny v daném pořadí, pokud data nejsou k dispozici v registrech, pozastaví se provádění všech instrukcí)
- O3CPU – simulace procesoru s pipeline, provádění instrukcí „out of order“ (instrukce jsou provedeny okamžitě po načtení dat do registrů procesoru, může docházet k přehazování instrukcí)

Model SimpleCPU poskytuje rychlou, ale méně přesnou simulaci a je tak vhodný například pro rychlý start operačního systému. Po startovací fázi simulace je možné model CPU přepnout na model „InOrder“ nebo na „O3CPU“. Všechny modely procesorů jsou nezávislé na instrukční sadě.

Simulátor GEM5 podporuje následující instrukční sady:

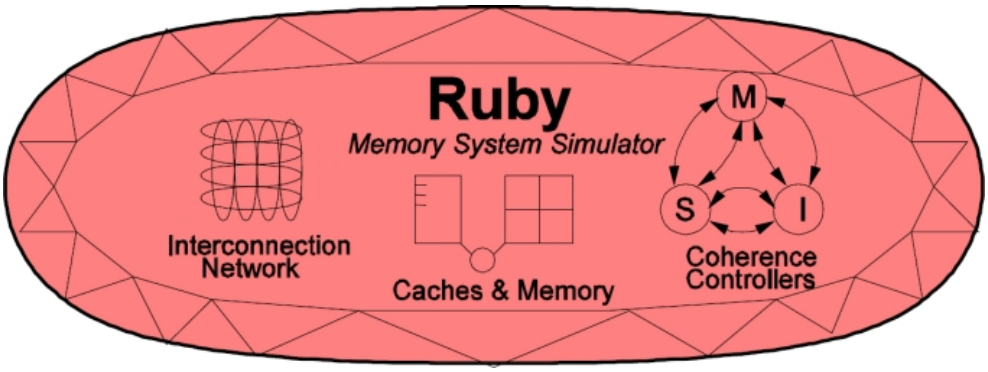
- Alpha
- ARM
- SPARC
- x86

Pro naše potřeby bude použita instrukční sada x86 a model procesoru „O3CPU“.

1.2 Ruby

Ruby je paměťový systém simulátoru GEM5, který je velmi flexibilní, konfigurovatelný a umožňuje vytvořit vlastní koherenční protokol s libovolnou hierarchii cache pamětí.

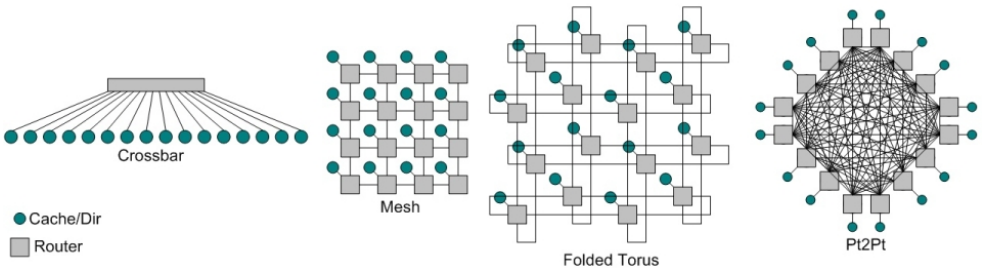
Popis paměťového systému Ruby je převážně založen na [2].



Obrázek 1.1: Ruby paměťový model (zdroj: [3])

Tento model se skládá z:

- propojovací sítě,
- sekvencerů,
- DMA řadičů,
- cache objektů,
- paměťového řadiče,
- SLICC a koherenčních protokolů.



Obrázek 1.2: Topologie v simulátoru GEM5 [3]

1.2.1 Propojovací sítě

Propojovací sítě jsou specifikované v jazyce Python. Ruby obsahuje dva modely „Simple“ a „Garnet“. Síťový model Garnet umožňuje detailnější nastavení sítě a jejích aktivních prvků, jeho nevýhodou je pomalejší simulace.

Simulátor GEM5 obsahuje několik předpřipravených topologií, které jsou znázorněné na obrázku 1.2 :

- Crossbar - Každý řadič je spojen s ostatními řadiči pomocí jednoho switche. Tato topologie je výchozí.
- Mesh (mřížka) - Počet routerů nebo switchů je roven počtu procesorů v systému. Každý aktivní prvek je připojen k jedné L1 cache, jedné L2 cache a k jednomu adresáři. Tato topologie vyžaduje aby se počet adresářů rovnal počtu procesorů.
- Point to point - Každý řadič cache a adresáře je přímo připojen ke všem ostatním řadičům přímým spojením.
- Torus (toroid) - Podobně jako mřížka s tím rozdílem že řádky a sloupce jsou realizovány pomocí kruhu.

Ruby rovněž umožňuje vytvořit vlastní topologii sítě a specifikovat několik parametrů sítě:

- latencem,
- váha - používané při směrování,
- propustnost.

Pro naše účely bude dostačovat model Simple s výchozí topologií (Crossbar). Více informací o rozdílech mezi modely paměťového systému Ruby naleznete v dokumentaci simulátoru GEM5 [3].

1.2.2 Sekvencer

Sekvencer je nejdůležitější částí paměťového systému Ruby. Spojuje jedno jádro procesoru s instrukční a datovou L1 cache pamětí. Každý požadavek na paměť musí projít přes sekvencer dvakrát – jednou sekvencer posílá žádost do L1 cache a podruhé při odpovědi L1 cache.

Hlavními úkoly sekvenceru jsou zpracování paměťových požadavků, alokace a kontrola prostředků (zajišťuje, aby nebyla překročena nastavená mez nevyřízených požadavků), přesvědčuje se, že atomické operace jsou správně obslouženy, kontroluje, zda koherenční protokol neuvázl, a předává odpovědi L1 cache jádru procesoru.

1.2.3 Cache objekty a strategie nahrazení

Cache objekty umožňují modelovat libovolnou paměť cache s omezeným stupněm asociativity. Každá instance modulu modeluje jednu paměťovou banku cache s nastavitelnou velikostí a stupněm asociativity. Paměť cache je reprezentována jako 2D paměťové pole.

V simulátoru GEM5 jsou dostupné dvě strategie výběru oběti Least Recently Used (LRU) a Pseudo Least Recently Used (Pseudo LRU). LRU ukládá informace o přístupu k datovému bloku a v případě nahrazení vybere datový blok, ke kterému nebylo nejdéle přistupováno. Pseudo LRU používá k výběru oběti binární strom – tato implementace je výrazně rychlejší, ale ne vždy vybere datový blok, ke kterému nebylo nejdéle přistupováno.

1.2.4 Řadič hlavní paměti

Řadič hlavní paměti modeluje jednokanálový řadič DDR pamětí. Umožňuje připojení libovolného počtu paměťových modulů a nastavení velkého množství parametrů. Je to jediný modul, který není implementován v jazyce SLICC, ale přímo v jazyce C++. Výchozí konfigurace modelu je paměť DDR-800. Více informací naleznete v [2].

1.2.5 DMA sekvencer

DMA sekvencer obsluhuje DMA požadavky. Systém může obsahovat několik DMA zařízení, ale obsahuje pouze jeden DMA sekvencer. Některé simulace, zejména v režimu *syscall emulation*, nevyžadují funkční DMA. Řadič DMA je implementován v jazyce SLICC.

1.3 SLICC

Zkratka SLICC znamená Specification Language for Implementing Cache Coherence (Jazyk pro implementaci cache koherence). Jedná se o jazyk, který je

použit pro vytvoření koherenčního protokolu a je realizován jako stavový automat. SLICC umožňuje specifikovat tyto stavové automaty a pomocí vstupních a výstupních portů je připojit do propojovací sítě.

SLICC se přeloží do jazyka C++ a při kompilaci simulátoru GEM5 se potřebné soubory připojí do výsledného programu. SLICC také umožňuje vygenerovat přechodové tabulky ve formátu HTML. Generování přechodových tabulek se provádí spuštěním následujícího příkazu v podadresáři *util*.

```
python slicc -H ~/HTML ../src/mem/protocol/MOESI.slicc
```

Řadič cache paměti se skládá z těchto částí:

- konečného stavového automatu,
- koherenčních zpráv,
- message bufferů a síťových portů,
- MSHR (Miss Status Handling Registry),
- akcí.

1.3.1 Konečný stavový automat

Stavový automat v jazyce SLICC se skládá ze stavů, přechodů a událostí. Události jsou vyvolány klíčovým slovem *trigger* a způsobují přechody mezi jednotlivými stavy automatů.

Stavy jsou definovány pomocí klíčového slova *state_declaration*. Při definici stavu se pomocí parametru *AccessPermission* určí přístupová oprávnění:

- Invalid - datový blok není dostupný,
- Read_Only - datový blok je dostupný jen pro čtení,
- Read_Write - datový blok je dostupný pro čtení a zápis,
- Busy - zaneprázdněn - požadavek bude pozastaven.

Přechody jsou definované pomocí klíčového slova *transition*, neprázdné množiny vstupních stavů, neprázdné množiny událostí a výstupního stavu. Tělo přechodu obsahuje seznam akcí, které se mají provést, pokud nastane daná událost. Celý přechod je proveden atomicky.

Na následující ukázce přechodu je znázorněn přechod L1 cache ze stavu IM do stavu M, který se aktivuje událostí *Data*. Řadič L1 cache nejprve zapíše data do cache paměti, poté upozorní sekvencer, dealokuje TBE záznam (Transaction Buffer Entry viz. sekce 1.3.4) a odebere příchozí zprávu ze vstupní fronty.

```
transition(IM, Data, M) {
   u_writeDataToCache;
   sx_store_hit;
   w_deallocateTBE;
   n_popResponseQueue;
}
```

1.3.2 Koherenční zprávy

Konečné automaty spolu komunikují pomocí zpráv. SLICC obsahuje tři typy zpráv:

- RubyRequest - zprávy od jader procesoru,
- zprávy pro komunikaci mezi stavovými automaty,
  - RequestMsg - pro požadavky,
  - ResponseMsg - pro odpovědi.

Na následující ukázce zdrojového kódu je uveden příklad definice zprávy RequestMsg.

```
structure(RequestMsg, desc="...", interface="NetworkMessage") {
  Address Addr,  desc="Physical address for this request";
  CoherenceRequestType Type, desc="Type of request (GetS, etc)";
  RubyAccessMode AccessMode,  desc="user/supervisor access type";
  MachineID Requestor, desc="What component request";
  NetDest Destination, desc="What components receive the request,
  includes MachineType and num";
  MessageSizeType MessageSize,  desc="size category of the message";
  DataBlock DataBlk, desc="Data for the cache line (if PUTX)";
  bool Dirty, default="false",  desc="Dirty bit";
  PrefetchBit Prefetch, desc="Is this a prefetch request";
  MachineID OriginalRequestorMachId,
                        desc="What component initiate request";
}
```

Na ukázce zprávy výše vidíme:

- Address - adresa datového bloku,
- CoherenceRequestType - typ požadavku (GetM, GetS, PutS, PutM apod.),
- RubyAccessMode - privilegovaný nebo neprivilegovaný přístup,
- MachineID - ID automatu, který vyslal požadavek,

- NetDest - množina adresátů implementována jako bitové pole, které obsahuje typ automatu a jeho číslo,
- MessageSizeType - velikost zprávy podle toho jestli zpráva obsahuje data,
- DataBlock - datový blok (u žádosti PUTM),
- Dirty bit - zda se datový blok liší od obsahu paměti,
- PrefetchBit - zda je požadavek vyslán prefetcherem.

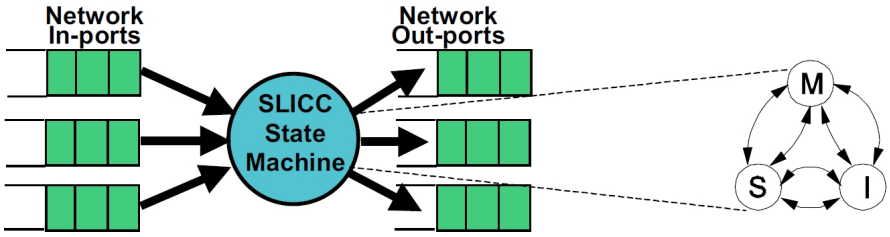
1.3.3 Message buffery a síťové porty

Message buffer funguje jako fronta pro příchozí a odchozí zprávy.

```
MessageBuffer requestFromL1Cache, network="To",
virtual_network="0", ordered="false", vnet_type="request";
```

V předchozí ukázce kódu je vidět několik parametrů pro deklaraci Message bufferu. Prvním parametrem „requestFromL1Cache“ je název message bufferu, dalším parametrem je „network“. Tento parametr může nabývat pouze dvou hodnot, „To“ pro odchozí zprávy a „From“ pro příchozí zprávy. Následuje parametr „virtual_network“, který definuje identifikátor virtuální sítě. Další parametr „ordered“ označuje zda příchozí zprávy mají být zpracovány v pořadí v jakém přišly. Poslední parametr „vnet_type“ je volitelný a označuje, jaký typ zpráv se posílá přes message buffer.

Síťové porty spojují řadiče vytvořené v jazyce SLICC s ostatními částmi Ruby propojovací sítě. Na obrázku 1.3 je vidět příklad připojení řadiče do sítě.



Obrázek 1.3: SLICC: připojení řadiče do sítě (zdroj: [3])

Na následující ukázce kódu je ukázka definice vstupního portu. První parametr určuje název portu, druhý typ přijímaných zpráv, další je název message bufferu a poslední parametr je „rank“, který určuje pořadí při probouzení bufferů po pozastavení zpráv. Tomuto parametru je potřeba věnovat pozornost, protože nesprávným nastavením tohoto parametru může velmi snadno dojít k uváznutí celého systému a jeho příčina se odhaluje velmi těžko.

```
in_port(requestNetwork_in, RequestMsg, requestToL1Cache, rank = 1) {
    if(requestNetwork_in.isReady()) {
      peek(requestNetwork_in, RequestMsg, block_on="Addr") {

        Entry cache_entry := getCacheEntry(in_msg.Addr);
        TBE tbe := L1_TBEs[in_msg.Addr];

        if (in_msg.Type == CoherenceRequestType:INV) {
          trigger(Event:Inv, in_msg.Addr, cache_entry, tbe);
        } else if (in_msg.Type == CoherenceRequestType:FWD_GETM) {
          trigger(Event:Fwd_GETM, in_msg.Addr, cache_entry, tbe);
        } else if (in_msg.Type == CoherenceRequestType:RECALL) {
          trigger(Event:Recall, in_msg.Addr, cache_entry, tbe);
        } else if (in_msg.Type == CoherenceRequestType:FWD_GETS) {
          trigger(Event:Fwd_GETS, in_msg.Addr, cache_entry, tbe);
        } else if (in_msg.Type == CoherenceRequestType:DOWNGRADE) {
          trigger(Event:Downgrade, in_msg.Addr, cache_entry, tbe);
        } else {
          error("Invalid forwarded request type");
        }
      }
    }
  }
```

Na předchozí ukázce dále vidíme přečtení zprávy pomocí funkce „peek“, získání cache bloku a TBE záznamu (viz. sekce 1.3.4). Dále vidíme roztřídění zpráv podle typu a následně vyvolání příslušné události pomocí funkce „trigger“.

1.3.4 Miss Status Handling Registers

MSHR jsou registry pro uchování informací o datovém bloku v tranzientním stavu. Obsahují adresu datového bloku, aktuální stav datového bloku v cache paměti, datový blok, příznak, zda jsou data modifikovaná, ID zařízení kam se mají poslat data, příznak, zda je požadavek vyvolán prefetcherem, a informace o počtu očekávaných potvrzení.

```
structure(TBE, desc="...") {
    Address Addr, desc="Physical Addr for this TBE";
    State TBEState, desc="Transient state";
    DataBlock DataBlk, desc="Buffer for the data block";
    bool Dirty, default="false", desc="data is dirty";
    MachineID Requestor, desc="Where to send data (FWD*)";
    bool isPrefetch, desc="Set if this was caused by a prefetch";
```

```
    int pendingAcks, default="0", desc="number of pending acks";
}
```

Zkratka TBE použitá v předchozí ukázce znamená Transaction Buffer Entry a je to pouze jiný název pro MSHR registry. TBE záznamy jsou seskupené do TBE tabulky, jedná se o pole indexované podle adresy datového bloku.

TBE záznam je nutné alokovat při událostech, které vyžadují přechod do tranzientního stavu. TBE záznamy mohou být použity pro počítání očekávaných potvrzení, uchování identifikátoru cache, kam se mají poslat data nebo potvrzení. Při přechodu do stabilního stavu je nutné TBE záznam dealokovat.

1.3.5 Akce

Akce jsou postupně vyvolávány v průběhu přechodů a jsou definované klíčovým slovem „action“. Akce jsou definované jako posloupnost operací, které určují, co se má s danou zprávou dělat.

```
action(a_issueRequest, "a", desc="Issue a request") {
   enqueue(requestNetwork_out, RequestMsg,
        latency=issue_latency) {
     out_msg.Address := address;
     out_msg.Type := CoherenceRequestType:GETX;
     out_msg.Requestor := machineID;
     out_msg.Destination.add(map_Address_to_Directory(address));
     out_msg.MessageSize := MessageSizeType:Control;
   }
 }
```

První argument obsahuje název akce, další obsahuje zkratku použitou v přechodových tabulkách a poslední argument obsahuje krátký popis akce.

Klíčové slovo „enqueue“ je použito pro odeslání zprávy (out_msg) na výstupní port, parametry určují, na který port se má zpráva poslat, jaký je její typ a doba, po které je možné danou zprávu přečíst. Pokud je povolena náhodnost doby čtení, je poslední parametr ignorován.

Pomocí akcí je možné také pozastavit nebo recyklovat zprávy. Tato funkce je velmi důležitá v případě, kdy se stavový automat nachází v tranzientním stavu a je potřeba příchozí požadavky pozastavit. Při zavolání funkce *stall_and_wait* se daná zpráva přesune z příchozího portu do postranní datové struktury, která je spojena s vstupním portem, tato zpráva již nebude znovu analyzována, dokud nebude explicitně probuzena funkcí *wakeUpBuffers* nebo *wakeUpAllBuffers*.

```
action(z_stallAndWaitMandatoryQueue, "\z",
   desc="recycle L1 request queue") {
     stall_and_wait(mandatoryQueue_in, address);
```

```
}

action(kd_wakeUpDependents, "kd", desc="wake-up dependents") {
     wakeUpBuffers(address);
}
```

Kapitola 2

MOESI protokol

V této kapitole si popíšeme MOESI protokol. Tento protokol slouží ke komunikaci mezi jádry procesoru, samotnými procesory a udržení koherence cache pamětí. Oproti klasickému MESI protokolu obsahuje navíc pátý stav „Owned“ tedy vlastník daného datového bloku, díky tomuto stavu není nutné zapisovat data do paměti při přechodu z modifikovaného do sdíleného stavu. Při návrhu MOESI protokolu bylo čerpáno z [4].

2.1 Specifikace

MOESI protokol je zlepšení stávajícího MESI protokolu. Jeho tvůrci se snažili omezit množství zápisů do paměti, při přechodu z modifikovaného stavu do sdíleného stavu. Tímto způsobem je značně urychleno sdílení mezi jednotlivými jádry procesoru, protože latence mezi cache paměťmi je výrazně nižší než mezi cache pamětí a hlavní pamětí.

2.1.1 Vlastnosti stavů

- Nejdůležitější vlastností je, zda je daný datový blok modifikován nebo ne. Stav je modifikovaný, pokud datový blok obsahuje modifikovaná data, která ještě nebyla zapsána do paměti.
- Další důležitou vlastností je možnost zápisu do daného datového bloku.
- Exkluzivní stav je také důležitý, protože umožňuje vlastníkovi rychle zapsat a nemusí čekat na ověření, zda jiný řadič cache neobsahuje požadovaný datový blok v platném stavu.

Nyní si zde uvedeme základní stavy protokolu MOESI. Popis tranzientních stavů, které utvářejí celý automat je uveden v sekci 3.4.2.

- Invalid - neplatný stav znamená, že daný řadič nemá správná data pro daný datový blok. Tento stav je zároveň startovací.

- Shared - sdílený stav znamená, že daný řadič má k dispozici platný datový blok, který může být sdílen s ostatními řadiči cache. To znamená, že daný blok je určen pouze ke čtení.

- Exclusive - exkluzivní stav je přiřazen datovému bloku, který není sdílený s jinými řadiči cache. Datový blok je nemodifikovaný ale s možností tichého přechodu do modifikovaného stavu.

- Exclusively Shared - datový blok je ve stavu Exclusive, ale je sdílený v nižší úrovni hierarchie cache. Tento stav je zaveden pro rychlé vyřízení požadavku na zápis, protože se nemusí komunikovat s cache vyšší úrovně.

- Owned - vlastník, datový blok označený tímto stavem je modifikován a zároveň může být sdílen řadiči cache nižší úrovně. Díky tomuto stavu se značně urychluje přechod z modifikovaného stavu, protože se modifikovaný datový blok nemusí zapisovat do paměti.

- Modified - modifikovaný stav, datový blok je modifikován a umožňuje čtení a zápis. Všechny ostatní řadiče cache jsou v neplatném stavu.

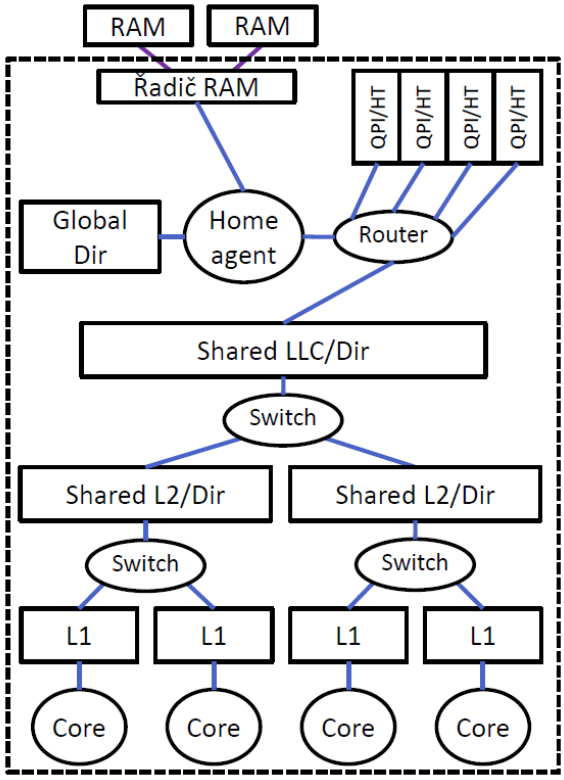
V tabulce 2.1 je uveden přehled vlastností jednotlivých stavů MOESI protokolu. „Tichý“ přechod znamená přechod do jiného stavu bez upozornění automatu vyšší úrovně.

	Modifikovaná	Unikátní	Zapisovatelná	„Tichý“ přechod
Modified	Ano	Ano	Ano	O
Owned	Ano	Ano	Ano	-
Exclusive	Ne	Ano	Ano	MS
Shared	Obojí	Ne	Ne	I
Invalid	Ne	Ne	Ne	-

Tabulka 2.1: Vlastnosti stavů

2.1.2 Hierarchie sdílených pamětí cache

Hlavním cílem této práce je vytvořit MOESI protokol optimalizovaný pro inkluzivní hierarchii cache pamětí uvedenou na obrázku 2.1. Sdílená L2 cache umožňuje rychlou výměnu datových bloků mezi procesory připojenými k dané L2 cache.



Obrázek 2.1: Hierarchie pamětí cache pro čtyřjádrový procesor

Hierarchie komunikujících automatů:

- Home agent - registruje v globálním adresáři stav a umístění kopií cache bloků z paměti připojené k tomuto procesoru. Tento automat předává požadavky čtení nebo zápisu řadiči paměti.
- Sdílená LLC registruje v přidruženém adresáři stav a umístění cache bloků na celém procesoru a může komunikovat s Home agenty na všech procesorech.
- Sdílená cache registruje v přidruženém adresáři stav a umístění cache v připojených cache nižší úrovně.
- L1 cache je rozdělena na instrukční a datovou část, které obsluhují požadavky příslušného jádra procesoru (Load, Store, Ifetch, Flush aj.)

V tabulce 2.2 jsou znázorněné povolené kombinace stavů vyššího a nižšího automatu v hierarchii. V každé buňce tabulky je na prvním řádku uveden stav vyššího automatu, na dalším řádku je označení zda se datový blok může nacházet ve více cache nižší úrovně (značky ↙ a ↘ pro více cache nižší úrovně,

značka ↓ označuje, že daný automat může být pouze jeden). Na posledním řádku každé buňky v tabulce je uveden stav, v jakém se může daný datový blok nacházet. Pokud je poslední řádek v buňce prázdný znamená to, že se daný datový blok nenachází v automatu nižší úrovně.

Stav vyšší úrovně	Stav nižší úrovně					
	S	ES	E	M	O	I
S	S ↙ ↘ S S					S ↓
ES	S ↙ ↘ S S					
E		E ↓ ES	E ↓ E	E ↓ M	E ↓ O	E ↓
O	O ↙ ↘ S S					O ↓
M				M ↓ M	M ↓ O	

Tabulka 2.2: Povolené kombinace stavů v hierarchii

Kapitola 3

Implementace

V této kapitole si popíšeme implementaci MOESI protokolu pro hierarchii sdílených pamětí cache. Použité propojení automatů je v sekci 3.1, zprávy jsou popsány v sekci 3.3 a mapování je popsáno v sekci 3.2. Implementace řadičů použitých v implementaci jsou popsány v následujících sekcích:

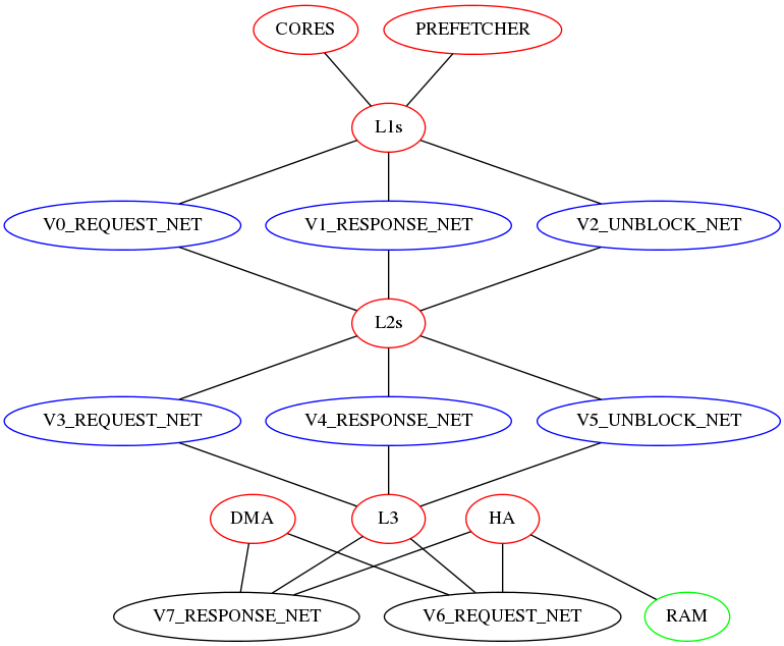
- L1 cache – sekce 3.4
- L2 cache – sekce 3.5
- L3 cache – sekce 3.6
- Home Agent – sekce 3.7
- Řadič DMA – sekce 3.8

Parametry použité pro simulaci jsou uvedeny v sekci 3.9.

3.1 Propojení automatů

Propojení mezi jednotlivými řadiči cache je realizováno pomocí tří virtuálních sítí. Ve skutečném procesoru tyto sítě jsou také virtuální.

Propojení automatů je znázorněno na obrázku 3.1.



Obrázek 3.1: Propojení hierarchie cache

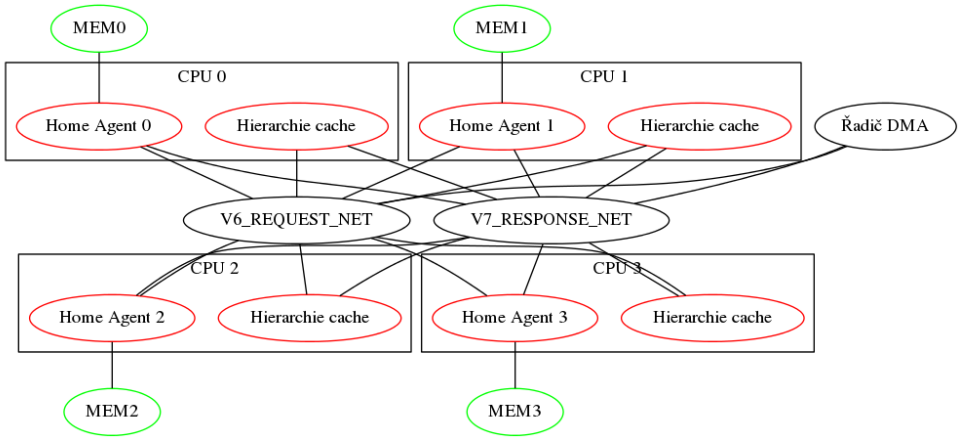
Každá L1 cache je rozdělena na instrukční a datovou část, obě části jsou řízené jedním řadičem. Ke každé L1 cache je připojeno jádro procesoru a prefetcher. L1 cache je dále připojena k L2 cache pomocí virtuálních sítí 0, 1 a 2. L2 cache je připojena k L3 cache pomocí virtuálních sítí 3, 4 a 5.

- Síť pro požadavky (REQUEST_NET) je použita pro posílání požadavků mezi dvěmi sousedními úrovněmi řadičů cache v rámci jednoho procesoru.
- Síť pro odpovědi (RESPONSE_NET) je použita pro posílání odpovědí mezi dvěmi sousedními úrovněmi řadičů cache v rámci jednoho procesoru. Tato síť je použita pro zasílání potvrzení a dat.
- Síť pro odblokování (UNBLOCK_NET) je použita pro posílání odblokovacích zpráv řadiči cache vyšší úrovně.

Pro propojení L3 cache a Home Agenta slouží další dvě virtuální sítě 6 a 7. Tyto sítě zároveň slouží k propojení více procesorů mezi sebou a slouží rovněž pro připojení DMA řadiče viz. obrázek 3.2.

- Síť pro požadvky - V6_REQUEST_NET
- Síť pro odpovědi - V7_RESPONSE_NET

Na obrázku 3.2 je znázorněno propojení čtyř procesorů a řadiče DMA operací pomocí virtuálních sítí 6 a 7.



Obrázek 3.2: Propojení více procesorů

Simulace propojení více procesorů není přesná, protože zde není nastavena latence mezi procesory, která je výrazně vyšší než mezi jednotlivými řadiči pamětí cache. Pro správnou simulaci by bylo potřeba definovat vlastní topologii, kde bude nastavena latence každému portu nebo použít síťový model „Garnet“.

3.2 Mapování adres do segmentů L2 a L3 cache

Každá L2 cache je rozdělena do dvou segmentů a každá L3 cache je rozdělena do osmi segmentů. Pro mapování adres do cache segmentů je využita funkce „mapAddressToRange“. Na následující ukázce je znázorněno volání funkce pro mapování do L2 cache:

```
mapAddressToRange(address, MachineType:L2Cache,
      l2_select_low_bit, l2_select_num_bits, intToID(l2_index))
```

Tato funkce vyžaduje pět parametrů. První je adresa datového bloku, druhý je typ automatu, třetí je počet bitů použitých pro adresaci jednotlivých bytů v datovém bloku (v našem případě je velikost bloku 64B, tento parametr tedy bude 6). Čtvrtým parametrem je počet bitů, podle kterých se budou adresy rozdělovat mezi jednotlivé segmenty cache (v našem případě 1 bit pro L2 cache a 3 bity pro L3 cache). Posledním parametrem je index klastru, tedy index všech segmentů sdružených do jedné cache.

3.3 Typy zpráv

Žádosti používané v naší implementaci jsou uvedeny v následující tabulce 3.1.

Typ	Popis	Data
GETS	Získej data pro čtení	Ne
GETM	Získej data pro zápis	Ne
DOWNGRADE	Přejdi do stavu S	Ne
INV	Zneplatni data (odpověď žadateli)	Ne
RECALL	Zneplatni data (odpověď odesílateli)	Ne
PUTM	Žádost o zneplatnění	Ano
DMA_READ	DMA žádost o čtení	Ne
DMA_WRITE	DMA žádost o zápis	Ano

Tabulka 3.1: Přehled žádostí

Odpovědi používané v naší implementaci jsou uvedeny v následující tabulce 3.2.

Typ	Popis	Data
MEMORY_ACK	Potvrzení z hlavní paměti	Ne
MEMORY_DATA	Data z hlavní paměti	Ano
DATA	Data	Ano
DATA_DIR	Data se seznamem sdílejících	Ano
DATA_EXCLUSIVE	Exklusivní data (s právem zápisu)	Ano
DATA_UNBLOCK	Data s odblokováním	Ano
ACK	Potvrzení	Ne
WB_ACK	Potvrzení zápisu	Ne
UNBLOCK	Odblokování	Ne

Tabulka 3.2: Přehled odpovědí

3.4 L1 cache

Každá L1 cache je rozdělena na instrukční a datovou část. Řadič L1 cache je připojen k instrukční a k datové cache paměti, dále je připojen k jádru procesoru (MandatoryQueue), prefetcheru (OptionalQueue) a k L2 cache.

Implementace řadiče L1 cache vychází z existujícího MESI protokolu. Zde bylo potřeba upravit chování automatu pro potřeby MOESI protokolu – doplnit stavy, přechody, události a akce spojené s přeposíláním dat mezi řadiči L1 cache apod.

3.4.1 L1 cache – stabilní stavy

L1 cache má čtyři stabilní stavy. Stav „Owned“ zde není potřeba, protože adresář ve vyšší úrovni cache ví, že tato cache je vlastníkem daného datového bloku. Stabilní stavy jsou M, E, S, I a jejich popis je v tabulce 3.3

Stav	Popis
M	Modifikovaná data
E	Nemodifikovaná data, tato cache je jediným vlastníkem
S	Nemodifikovaná data, data jsou sdílena s jinými cache
I	Data jsou neplatná nebo nejsou obsažena v této cache

Tabulka 3.3: Popis stabilních stavů L1 cache

Autoři MESI protokolu používají navíc stav „Not Present“ jako startovací. Je použit pouze při startu simulace a nikdy se do tohoto stavu nevrátí. Proto jsme se rozhodli tento stav sloučit se stavem „Invalid“.

3.4.2 L1 cache – Tranzientní stavy

Tranzientní stavy slouží pro zablokování automatu do okamžiku, než přijde odpověď na vyslaný požadavek. Popis tranzientních stavů je v tabulce 3.4.

Stav	Popis
IS	Neplatná data, odeslán požadavek GETS do L2 cache
ISD	Neplatná data, odeslán požadavek GETS do L2 cache, přišel požadavek Downgrade před daty
IM	Neplatná data, odeslán požadavek GETM do L2 cache
IIM	Odeslán požadavek GETM, přišla odpověď DATA_DIR, čeká na potvrzení od sdílejících L1 cache
SM	Přechod ze sdíleného do modifikovaného stavu, odeslán požadavek GETM, čeká na odpověď
SSM	Přišla odpověď DATA_DIR, čeká na potvrzení od sdílejících L1 cache
IS_I	Neplatná data, odeslán požadavek GETS do L2 cache, přišla žádost o zneplatnění (Inv/Recall)
MI	Odeslán požadavek PUTM, čeká na potvrzení o zápisu
SI	Odeslán požadavek PUTS, čeká na potvrzení
SI_F	Flush - vyslání žádosti PUTS do L2, čeká na ACK
MI_F	Flush - vyslání žádosti PUTM do L2, čeká na WBACK

Tabulka 3.4: Popis tranzientních stavů L1 cache

V tabulce 3.5 jsou popsány tranzientní stavy spojené s prefetcherem.

Stav	Popis
PF_IS	Neplatná data, odeslán požadavek GETS do L2 cache
PF_ISD	Neplatná data, odeslán požadavek GETS do L2 cache, přišla žádost DOWNGRADE
PF_IM	Neplatná data, odeslán požadavek GETM do L2 cache
PF_IIM	Odeslán požadavek GETM, přišla odpověď DATA_DIR, čeká na potvrzení od sdílejících L1 cache
PF_IS_I	Neplatná data, odeslán požadavek GETS do L2 cache, přišla žádost o invalidaci (Inv)
PF_ISR_I	Neplatná data, odeslán požadavek GETS do L2 cache, přišla žádost o zneplatnění (Recall)

Tabulka 3.5: Popis tranzientních stavů L1 cache - prefetch

3.4.3 L1 cache – události

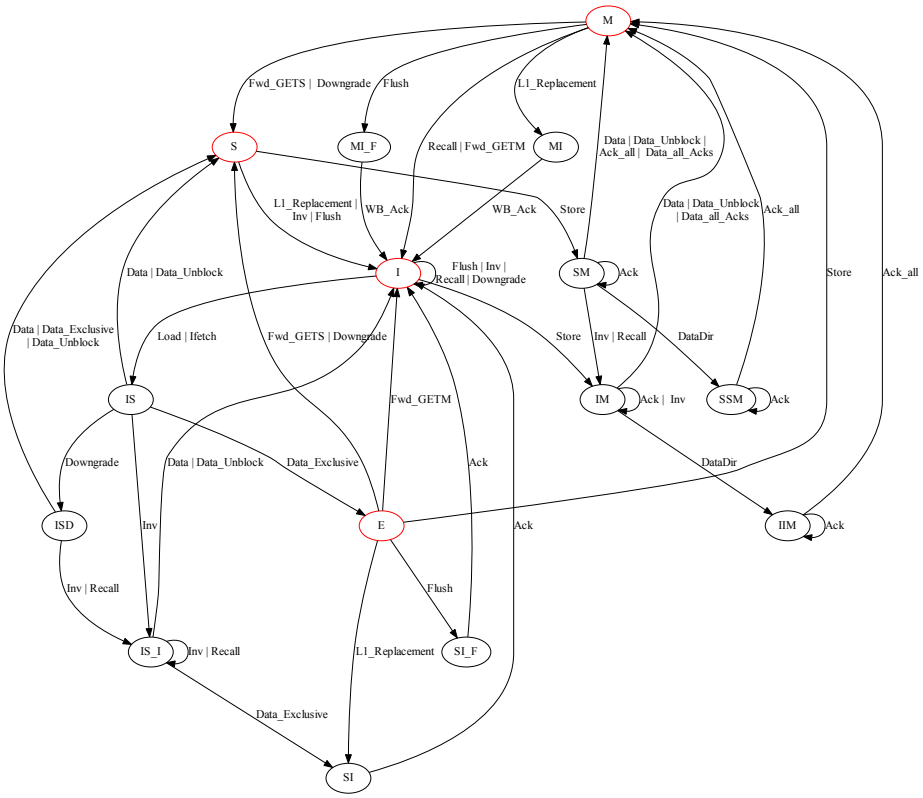
Události, které mohou nastat v L1 cache, jsou popsané v tabulce 3.6. Zdrojem událostí může být: jádro procesoru, prefetcher, L2 cache, jiná L1 cache nebo vlastní řadič.

Událost	Iniciátor	Popis
Load	CPU	Žádost o čtení z datové cache
Ifetch	CPU	Žádost o čtení z instrukční cache
Store	CPU	Žádost o zápis
Flush	CPU	Žádost o uvolnění cache
Inv	L2 Cache	Žádost o zneplatnění z jiné L1 Cache
Recall	L2 Cache	Žádost o zneplatnění z L2 Cache
Downgrade	L2 Cache	Žádost o přechod do stavu S
L1_Replacement	L1 Cache	Uvolnění místa v cache
PF_L1_Replacement	L1 Cache	Uvolnění místa v cache pro prefetch
Fwd_GETM	L2 Cache	Žádost o přeposlání dat pro zápis jiné L1 Cache, přechod do neplatného stavu
Fwd_GETS	L2 Cache	Žádost o přeposlání dat pro čtení jiné L1 Cache, přechod do sdíleného stavu
Data	L2 Cache	Data pro procesor
Data_Exclusive	L2 Cache	Exkluzivní data pro procesor
Data_Unblock	L2 Cache	Data pro procesor, odešli Unblock do L2 Cache
DataDir	L2 Cache	Data s adresářem
Data_all_acks	L2 Cache	Data pro procesor, se všemi potvrzeními
Ack	L1/L2 Cache	Potvrzení
Ack_all	L1/L2 Cache	Všechna potvrzení
WB_Ack	L2 Cache	Potvrzení o zápisu dat do L2 Cache
PF_Load	Prefetcher	Žádost o přednačtení dat do datové L1 Cache
PF_Ifetch	Prefetcher	Žádost o přednačtení dat do instrukční L1 Cache
PF_Store	Prefetcher	Žádost o přednačtení dat do datové L1 Cache pro zápis

Tabulka 3.6: L1 Cache – popis událostí

3.4.4 L1 cache – řadič

Na obrázku 3.3 je znázorněn přechodový diagram řadiče L1 cache bez stavů potřebných pro obsluhu požadavků prefetcheru, kompletní obrázek lze nalézt v příloze nebo na přiloženém CD.



Obrázek 3.3: Přechodový diagram řadiče L1 cache, stabilní stavy jsou vyznačeny červeně

3.5 L2 cache

L2 cache je sdílena dvěma L1 cache a je rozdělena do dvou segmentů. Řadič L2 cache obsahuje integrovaný adresář, který registruje řadiče L1 cache, které jsou připojeny ke konkrétní L2 cache.

3.5.1 L2 cache – propojení

L2 cache je připojena k virtuálním sítím 0, 1, 2 které spojují L2 cache s L1 cache. L2 cache je dále připojena pomocí virtuálních sítí 3, 4, 5 k L3 cache.

3.5.2 L2 cache – stabilní stavy

L2 cache má šest stabilních stavů. Oproti klasickému MESI protokolu přibyly navíc stavy O a ES. Stav O (Owned) umožňuje sdílení modifikovaného datového bloku cache paměťmi nižší úrovně, aniž by se data zapisovala do hlavní paměti. Stav ES (Exclusively Shared) umožňuje sdílení datového bloku v nižších úrovních cache, přičemž L2 cache si zachová exkluzivitu, to znamená, že pokud některý procesor pod L2 cache zažádá o zápis do daného datového bloku, L2 cache může tiše přejít do modifikovaného stavu.

Stabilní stavy jsou popsány v tabulce 3.7

Stav	Popis
M	Modifikovaná data v některé cache nižší úrovně
O	Modifikovaná data v L2 cache, data mohou být sdílena v nižší úrovni cache
E	Exkluzivní data v L2 cache
ES	Exkluzivní data v L2 cache, data mohou být sdílena v nižší úrovni
S	Sdílená data v L2 cache
I	Data nejsou v L2 cache

Tabulka 3.7: Popis stabilních stavů L2 cache

3.5.3 L2 cache – tranzientní stavy

Popis tranzientních stavů řadiče L2 cache je uveden v tabulkách 3.8 a 3.9.

Stav	Popis
IS	Přišla žádost GetS, čeká na odpověď z L3 cache
IS_II	Před daty přišla žádost o zneplatnění, odeslán Recall do nižší úrovně, čeká na potvrzení od všech žadatelů
IS_I	Potvrzení od všech žadatelů, odesláno potvrzení L3 nebo L2 cache
I_II	Přišla všechna potvrzení nebo data, data odeslaná všem žadatelům, čeká na všechna potvrzení
IM	Přišla žádost GetM, čeká na odpověď od L3 cache
IIM	Přišla data a adresář, čeká na potvrzení od sdílejících L2 cache
EI	Potřeba uvolnit místo v L2 cache, nikdo z nižších úrovní cache nesdílí data, odeslána žádost PutS nebo PutM, čeká na potvrzení od L3 cache
ERI	Potřeba uvolnit místo v L2 cache, odeslána žádost Recall do cache nižší úrovně, čeká na data nebo potvrzení
ERCI	Přišla žádost Recall, odeslána žádost Recall, čeká na data nebo potvrzení
EE	Přišla žádost GetS, odeslána žádost Fwd_GetS, čeká na data nebo potvrzení
EU	Přišlo odblokování před daty, čeká na data
EM	Přišla žádost GetM, odeslána žádost Fwd_GetM, čeká na potvrzení nebo odblokování
EFS	Přišla žádost Fwd_GetS, přeposílá Fwd_GetS do nižší úrovně cache, čeká na data nebo potvrzení
EFM	Přišla žádost Fwd_GetM, odeslán Recall do nižší úrovně, čeká na data nebo potvrzení
SI	Potřeba uvolnit místo v L2 cache, odeslán Recall sdílejícím cache nižší úrovně, čeká na všechna potvrzení
SIN	Přišla žádost o zneplatnění, odeslán Recall do cache nižší úrovně, čeká na všechna potvrzení
SIM	Přišla žádost o zneplatnění (ve stavu SM), odeslán Recall do cache nižší úrovně
SM	Přišla žádost GetM, odeslána žádost Inv do cache nižší úrovně, odeslán požadavek GetM do L3 cache, čeká na data
SSM	Přišla data a adresář, čeká na potvrzení od ostatních L2 cache

Tabulka 3.8: Popis tranzientních stavů L2 cache

Stav	Popis
ORCI	Přišla žádost Recall, odeslán Recall do nižší úrovně cache, čeká na potvrzení od sdílejících z nižší úrovně
ORI	Potřeba uvolnit místo v L2 cache, odeslán Recal do nižší úrovně, čeká na potvrzení od sdílejících z nižší úrovně
OFM	Přišla žádost Fwd_GetM, odeslán Recall do nižší úrovně, čeká na potvrzení z nižší úrovně
MM	Přišla žádost GetM, odeslána žádost Fwd_GetM nebo odeslána data, čeká na odblokování
MO	Přišla žádost GetS, odeslána žádost Fwd_GetS, čeká na data
MRCI	Přišla žádost Recall, odeslán Recall, čeká na data z nižší úrovně
MRI	Potřeba uvolnit místo v L2 cache, odeslán Recall do nižší úrovně, čeká na data
MI	Odeslána žádost PutM, čeká na potvrzení o zápisu od L3 cache
MFS	Přišla žádost Fwd_GetS, přeposlána žádost Fwd_GetS, čeká na data
MFM	Přišla žádost Fwd_GetM, odeslán Recall, čeká na data
MS	Přišla žádost Downgrade, přeposlána žádost Downgrade, čeká na data z L1 cache

Tabulka 3.9: Popis tranzientních stavů L2 cache

3.5.4 L2 cache – události

Události v L2 cache jsou popsány v tabulkách 3.10 a 3.11. Události „Ack“ a „Ack3“ se liší tím, ze které virtuální sítě přicházejí. „Ack“ přichází z L1 cache a Ack3 přichází z L2 nebo L3 cache. Události s příponou „_NS“ označují, že datový blok není platný v žádné z připojených L1 cache.

Stav	Iniciátor	Popis
GetS	L1 cache	Žádost o data ke čtení
GetSE	L1 cache	Žádost o data ke čtení, v žádné z cache nižší úrovně není daný datový blok
GetM	L1 cache	Žádost o data pro zápis
GetM_NS	L1 cache	Žádost o data pro zápis, v žádné z cache nižší úrovně není daný datový blok
Downgrade	L3 cache	Přechod do stavu S
Downgrade_NS	L3 cache	Přechod do stavu S, v žádné z cache nižší úrovně není daný datový blok
Fwd_GetS	L3 cache	Žádost o přeposlání dat jiné L2 cache pro čtení
Fwd_GetS_NS	L3 cache	Žádost o přeposlání dat jiné L2 cache, v žádné z cache nižší úrovně není daný datový blok
Fwd_GetM	L3 cache	Žádost o přeposlání dat jiné L2 cache pro zápis
Fwd_GetM_NS	L3 cache	Žádost o přeposlání dat jiné L2 cache pro zápis, v žádné z cache nižší úrovně není daný datový blok
PutM	L1 cache	Žádost o zápis dat do paměti
PutMD	L1 cache	Totéž jako PutM, ale data se nezapisují, protože nemusejí být aktuální
Replacement	L2 cache	Nahrazení datového bloku
ReplacementNS	L2 cache	Nahrazení datového bloku, nikdo nesdílí
ReplacementNS_Dirty	L2 cache	Nahrazení datového bloku, nikdo nesdílí, modifikovaná data

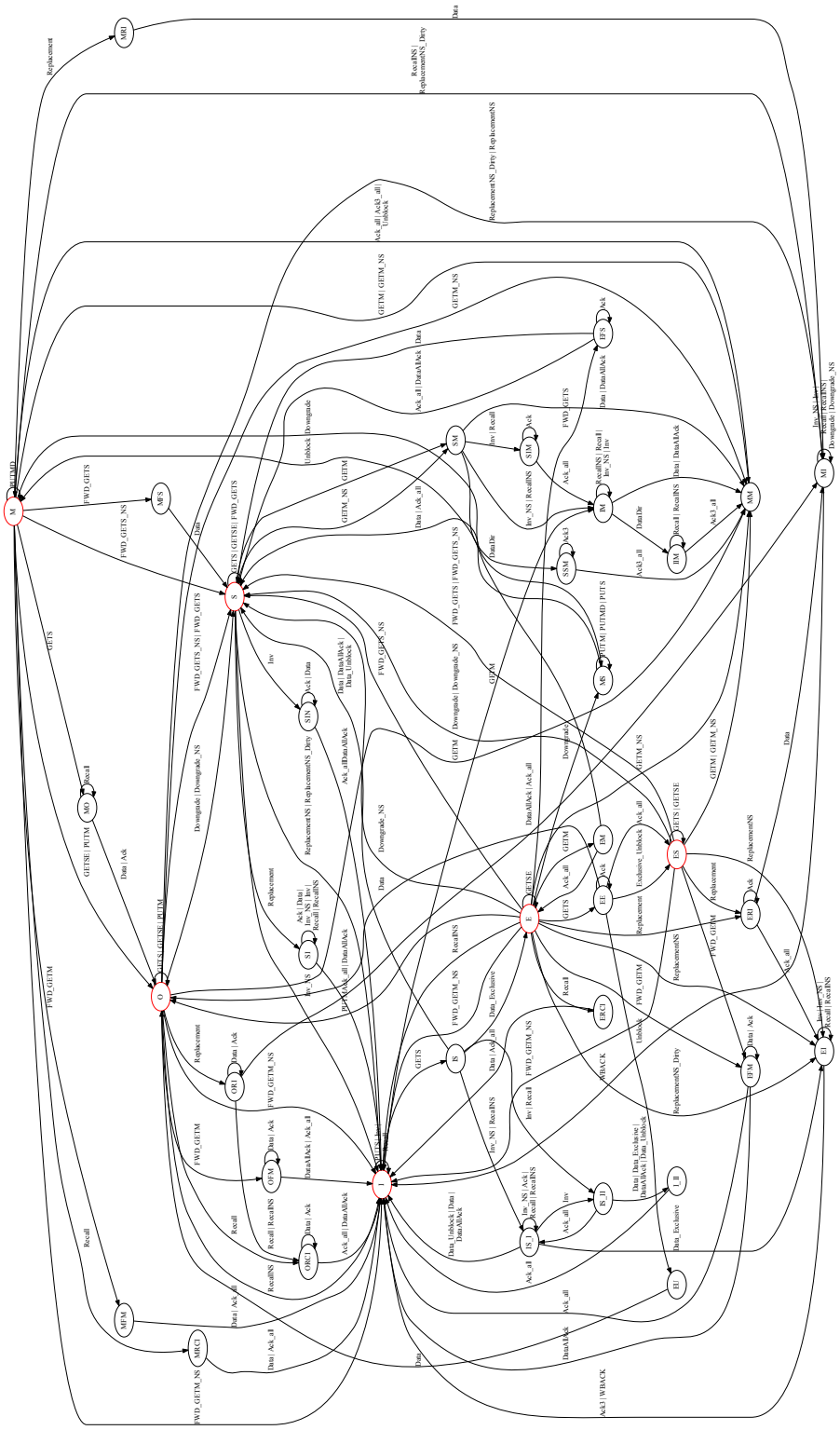
Tabulka 3.10: Popis událostí L2 cache

Stav	Iniciátor	Popis
Data	L1 / L2 cache	Data z L1 nebo L2 cache
DataDir	L3 cache	Data a adresář z L3 cache
Data_Exclusive	L3 cache	Exkluzivní data z L3 cache
Data_all_acks	L3 cache	Data z L3 cache, všechna potvrzení
Data_Unblock	L2 cache	Data a odblokování z L2 cache
WBAck	L3 cache	Potvrzení o zápisu
Ack3	L2 / L3 cache	Potvrzení
Ack3_All	L2 / L3 cache	Všechna potvrzení
Ack	L1 cache	Potvrzení
Ack_All	L1 cache	Všechna potvrzení
Unblock	L1 cache	Odblokování
Exclusive_Unblock	L1 cache	Odblokování
Inv	L3 cache	Invalidace z L3 cache
Inv_NS	L3 cache	Invalidace z L3 cache, nikdo nesdílí
Recall	L3 cache	Zneplatnění z L3 cache
RecallNS	L3 cache	Zneplatnění z L3 cache, nikdo nesdílí

Tabulka 3.11: Popis událostí L2 cache – pokračování

3.5.5 L2 cache – řadič

Přechodový diagram řadiče L2 cache je znázorněn na obrázku 3.4. Diagram se rovněž nachází na přiloženém CD.



Obrázek 3.4: Přechodový diagram řadiče L2 cache

3.6 L3 cache

L3 cache je sdílena všemi L2 cache v rámci jednoho čipu. L3 cache je také segmentovaná a počet segmentů se odvíjí od počtu jader daného procesoru. L3 cache rovněž obsahuje vestavěný adresář, který registruje všechny řadiče L2 cache na čipu s platnou kopií daného datového bloku.

3.6.1 L3 cache – propojení

L3 cache je připojena k virtuálním sítím 6 a 7, které spojují L3 cache, Home Agenty a řadiče DMA. L3 cache je dále připojena pomocí virtuálních sítí 3, 4, 5 k L2 cache.

3.6.2 L3 cache – stabilní stavy

L3 cache má šest stabilních stavů. Oproti klasickému MESI protokolu přibyly navíc stavy O a ES. Stav O (Owned) umožňuje sdílení modifikovaného datového bloku cache paměťmi nižší úrovně, aniž by se data zapisovala do hlavní paměti. Stav ES (Exclusively Shared) umožňuje sdílení datového bloku v nižších úrovních cache, přičemž L3 cache si zachová exkluzivitu, to znamená, že pokud některý procesor pod L3 cache zažádá o zápis do daného datového bloku, L3 cache může tiše přejít do modifikovaného stavu.

Stabilní stavy jsou popsány v tabulce 3.12

Stav	Popis
M	Modifikovaná data v některé cache nižší úrovně
O	Modifikovaná data v L3 cache, data mohou být sdílena v nižších úrovních cache
E	Exkluzivní data v L3 cache
ES	Exkluzivní data v L3 cache, data mohou být sdílena v nižších úrovních
S	Sdílená data v L3 cache
I	Data nejsou v L3 cache

Tabulka 3.12: Popis stabilních stavů L3 cache

3.6.3 L3 cache – tranzientní stavy

Tranzientní stavy jsou uvedeny v tabulkách 3.13 a 3.14. Ve stavu IS je možné přijímat další požadavky GetS. Žadatelé se zaznamenávají do TBE záznamu a po obdržení dat z paměti se data odešlou všem žadatelům najednou.

Stav	Popis
IS	Přišla žádost GetS, čeká na data z paměti, může přijímat další požadavky GetS
IS_II	Před daty přišla žádost o zneplatnění, odeslán Recall do nižší úrovně, čeká na potvrzení od všech žadatelů
IS_I	Potvrzení od všech žadatelů, odesláno potvrzení Home Agentu nebo L3 cache
I_II	Přišla potvrzení nebo data, data odeslána všem žadatelům, čeká na všechna potvrzení
IM	Přišla žádost GetM, čeká na data z paměti
IIM	Přišla data a adresář, čeká na potvrzení od sdílejících L3 cache
EI	Potřeba uvolnit místo v L3 cache, žádná z nižších úrovní cache nesdílí data, odeslána žádost PutS nebo PutM, čeká na potvrzení od Home Agenta
ERI	Potřeba uvolnit místo v L3 cache, odeslána žádost Recall do cache nižší úrovně, čeká na data nebo potvrzení
ERCI	Přišla žádost Recall, odeslána žádost Recall, čeká na data nebo potvrzení

Tabulka 3.13: Popis tranzientních stavů L3 cache

Stav	Popis
EE	Přišla žádost GetS, odeslána žádost Fwd_GetS, čeká na data nebo potvrzení
EM	Přišla žádost GetM, odeslána žádost Fwd_GetM, čeká na potvrzení nebo odblokování
EFS	Přišla žádost Fwd_GetS, odeslán Downgrade do nižší úrovně cache, čeká na data nebo potvrzení
EFM	Přišla žádost Fwd_GetM, odeslán Recall do nižší úrovně, čeká na data nebo potvrzení
EU	Přišlo odblokování před daty, čeká na data
SI	Potřeba uvolnit místo v L3 cache, odeslán Recall sdílejícím cache nižší úrovně, čeká na všechna potvrzení
SII	Všechna potvrzení z L2 cache, odeslána žádost PutS, čeká na potvrzení od Home Agenta
SIN	Přišla žádost o zneplatnění, odeslán Recall do cache nižší úrovně, čeká na všechna potvrzení
SIM	Přišla žádost o zneplatnění (ve stavu SM), odeslán Recall do cache nižší úrovně
SM	Přišla žádost GetM, odeslána žádost Inv do cache nižší úrovně, odeslán požadavek GetM Home Agentu, čeká na data
SSM	Přišla data a adresář, čeká na potvrzení od ostatních L3 cache
ORCI	Přišla žádost Recall, odeslán Recall do nižší úrovně cache, čeká na potvrzení od sdílejících z nižší úrovně
ORI	Potřeba uvolnit místo v L3 cache, odeslán Recal do nižší úrovně, čeká na potvrzení od sdílejících z nižší úrovně
OFM	Přišla žádost Fwd_GetM, odeslán Recall do nižší úrovně, čeká na potvrzení z nižší úrovně
MM	Přišla žádost GetM, odeslána žádost Fwd_GetM nebo odeslána data, čeká na odblokování
MO	Přišla žádost GetS, odeslána žádost Fwd_GetS, čeká na data
MRCI	Přišla žádost Recall, odeslán Recall, čeká na data z nižší úrovně
MRI	Potřeba uvolnit místo v L3 cache, odeslán Recall do nižší úrovně, čeká na data
MI	Odeslána žádost PutM, čeká na potvrzení o zápis od Home Agenta
MFS	Přišla žádost Fwd_GetS, odeslán Downgrade, čeká na data
MFM	Přišla žádost Fwd_GetM, odeslán Recall, čeká na data
UM	Odeslána data do nižší úrovně, čeká na odblokování

Tabulka 3.14: Popis tranzientních stavů L3 cache

3.6.4 L3 cache – události

Události v L3 cache jsou popsány v tabulkách 3.15 a 3.16. Události s příponou „_NS“ označují, že datový blok není platný v žádné z připojených L2 cache.

Stav	Iniciátor	Popis
GetS	L2 cache	Žádost o data ke čtení
GetSE	L2 cache	Žádost o data ke čtení, data nejsou v žádné L2 cache na čipu
GetM	L2 cache	Žádost o data pro zápis
GetM_NS	L2 cache	Žádost o data pro zápis, data nejsou v žádné L2 cache na čipu
Fwd_GetS	Home Agent	Žádost o přeposlání dat pro čtení jinému procesoru
Fwd_GetS_NS	Home Agent	Žádost o přeposlání dat pro čtení jinému procesoru, data nejsou v žádné L2 cache na čipu
Fwd_GetM	Home Agent	Žádost o přeposlání dat pro zápis jinému procesoru
Fwd_GetM_NS	Home Agent	Žádost o přeposlání dat pro zápis jinému procesoru
PutM	L2 cache	Žádost o zápis dat do paměti
PutMD	L2 cache	Totéž jako PutM, data se ale nezapisují protože nejsou aktuální
Replacement	L3 cache	Nahrazení datového bloku
ReplacementNS	L3 cache	Nahrazení datového bloku, data nejsou v žádné L2 cache na čipu
ReplacementNS_Dirty	L3 cache	Nahrazení datového bloku, data nejsou v žádné L2 cache na čipu, modifikovaná data

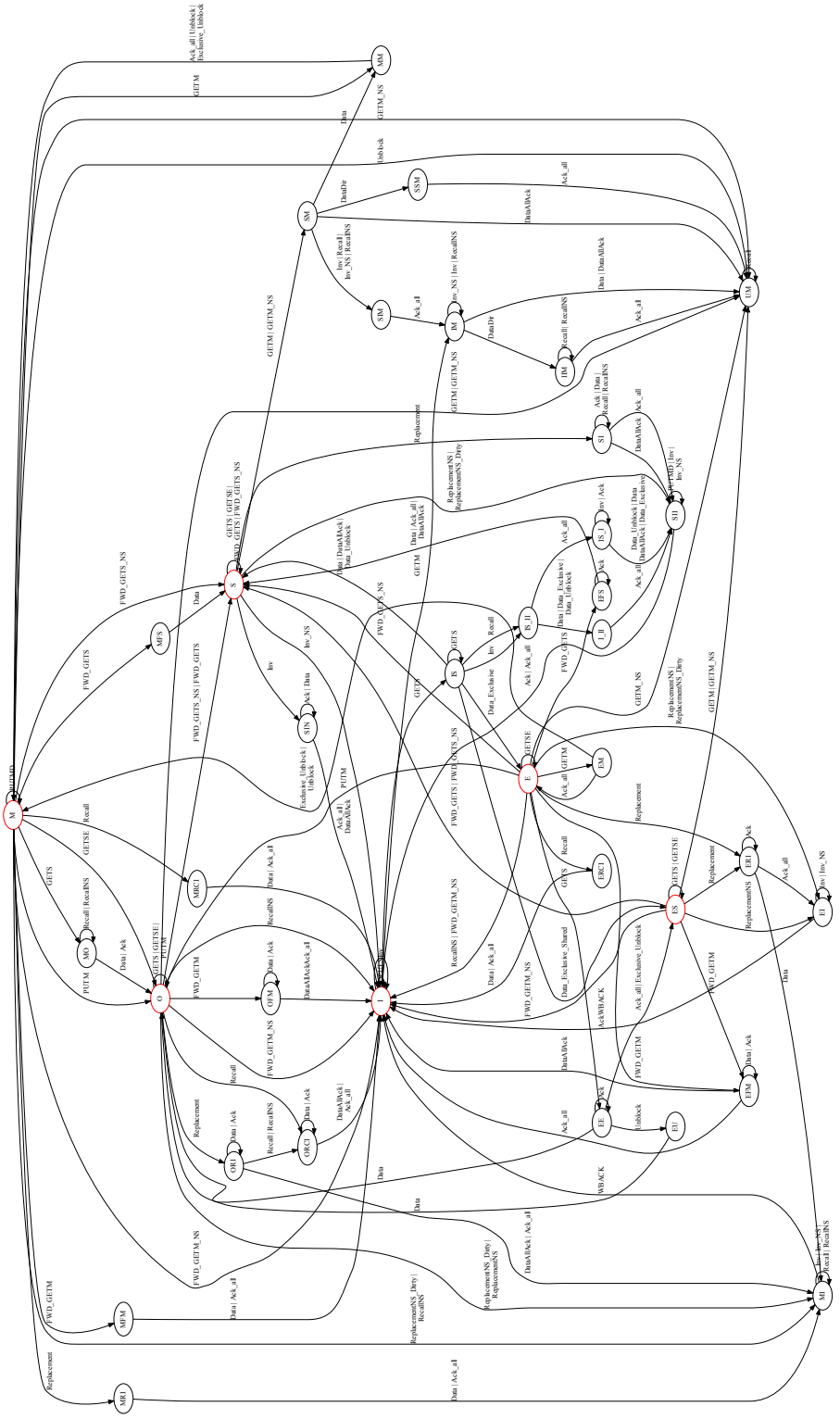
Tabulka 3.15: Popis událostí L3 cache

Stav	Iniciátor	Popis
Data	L2 / L3 cache	Data z L2 nebo L3 cache
DataDir	HA	Data s adresářem od Home Agenta
Data_Exclusive	HA	Exkluzivní data od Home Agenta
Data_Exclusive_Shared	HA	Exkluzivní data od Home Agenta
Data_all_acks	HA	Data od Home Agenta, všechna potvrzení
Data_Unblock	L3 cache	Data od jiné L3 cache včetně odblokování
WBAck	HA	Potvrzení o zápisu do paměti
Ack	HA / L2 / L3 cache	Potvrzení
Ack_All	L2 / L3 cache	Všechna potvrzení
Unblock	L2 cache	Odblokování
Exclusive_Unblock	L2 cache	Odblokování
Inv	HA	Invalidace
Inv_NS	HA	Invalidace, data nejsou v žádné L2 cache na čipu
Recall	HA	Zneplatnění
RecallNS	HA	Zneplatnění, data nejsou v žádné L2 cache na čipu

Tabulka 3.16: Popis událostí L3 cache – pokračování

3.6.5 L3 cache – řadič

Přechodový diagram řadiče L3 cache je znázorněn na obrázku 3.5. Diagram se rovněž nachází na přiloženém CD.



Obrázek 3.5: Přechodový diagram L3 cache

3.7 Home Agent

Home Agent registruje v globálním adresáři stav a umístění kopií datových bloků z hlavní paměti připojené ke konkrétnímu procesoru. Obsluhuje požadavky ze všech procesorů v systému a také obsluhuje požadavky z řadiče DMA.

Implementace vznikla modifikací adresáře z existujícího MOESI protokolu.

3.7.1 Home Agent – propojení

Home Agent je připojen k virtuálním sítím 6 a 7, které spojují L3 cache a ostatní Home Agenty. K Home Agentu je připojen řadič paměti.

3.7.2 Home Agent – stabilní stavy

Home Agent má čtyři stabilní stavy, které jsou popsány v tabulce 3.17

Stav	Popis
M	Modifikovaná data v jedné z L3 cache
E	Exkluzivní data v jedné z L3 cache
S	Sdílená data v jedné nebo více L3 cache
I	Data nejsou v žádné L3 cache

Tabulka 3.17: Popis stabilních stavů Home Agenta

3.7.3 Home Agent – tranzientní stavy

Popis tranzientních stavů Home Agenta je popsán v tabulce 3.18.

Stav	Popis
IE	Přišla žádost GetS, čeká na data z paměti
EI	Přišla žádost PutM, čeká na potvrzení zápisu z paměti
SS	Přišla žádost GetS, odeslán požadavek Fwd_GetS, čeká na potvrzení
SM	Přišla žádost GetM, čeká na data z paměti nebo na potvrzení z L3 cache
SA	Přišla žádost GetS, čeká na potvrzení nebo data z L3 cache
SD	Následuje po stavu SA, přišla data, čeká na potvrzení zápisu z paměti
IM	Přišla žádost GetS, čeká na data z paměti
MM	Přišla žádost GetM, čeká na potvrzení z L3 cache
MI	Přišla žádost PutM, čeká na potvrzení z paměti
MSA	Přišla žádost GetS, čeká na data z L3 cache
MDS	Navazuje na MSA, přišla data, čeká na potvrzení zápisu z L3 cache
MRPI	NEPOUŽITO REPLACEMENT
MRPDI	NEPOUŽITO REPLACEMENT
ID	Žádost řadiče DMA o čtení, čeká na data z paměti
ID_W	Žádost řadiče DMA o zápis, čeká na potvrzení zápisu z paměti
SDM	Žádost řadiče DMA o čtení, čeká na data z paměti
M_DRD	Žádost řadiče DMA o čtení, čeká na data nebo potvrzení z paměti
M_DRDI	Žádost řadiče DMA o čtení, čeká na potvrzení zápisu z paměti
M_DWR	Žádost řadiče DMA o zápis, čeká na potvrzení nebo data z L3 cache
M_DWRI	Žádost řadiče DMA o zápis, čeká na potvrzení zápisu z paměti

Tabulka 3.18: Popis tranzientních stavů Home Agenta

Stav SS je jediný tranzientní stav, který reaguje na žádosti GetS od jiných L3 cache. Při přijetí požadavku GetS ve stavu S nebo SS se Home Agent podívá do adresáře, zda je datový blok přítomen v lokální L3 cache pokud ano, pošle žádost Fwd_GetS lokální L3 cache, pokud datový blok není v lokální L3 cache, požadavek Fwd_GetS se pošle libovolné L3 cache s daným datovým blokem.

3.7.4 Home Agent – události

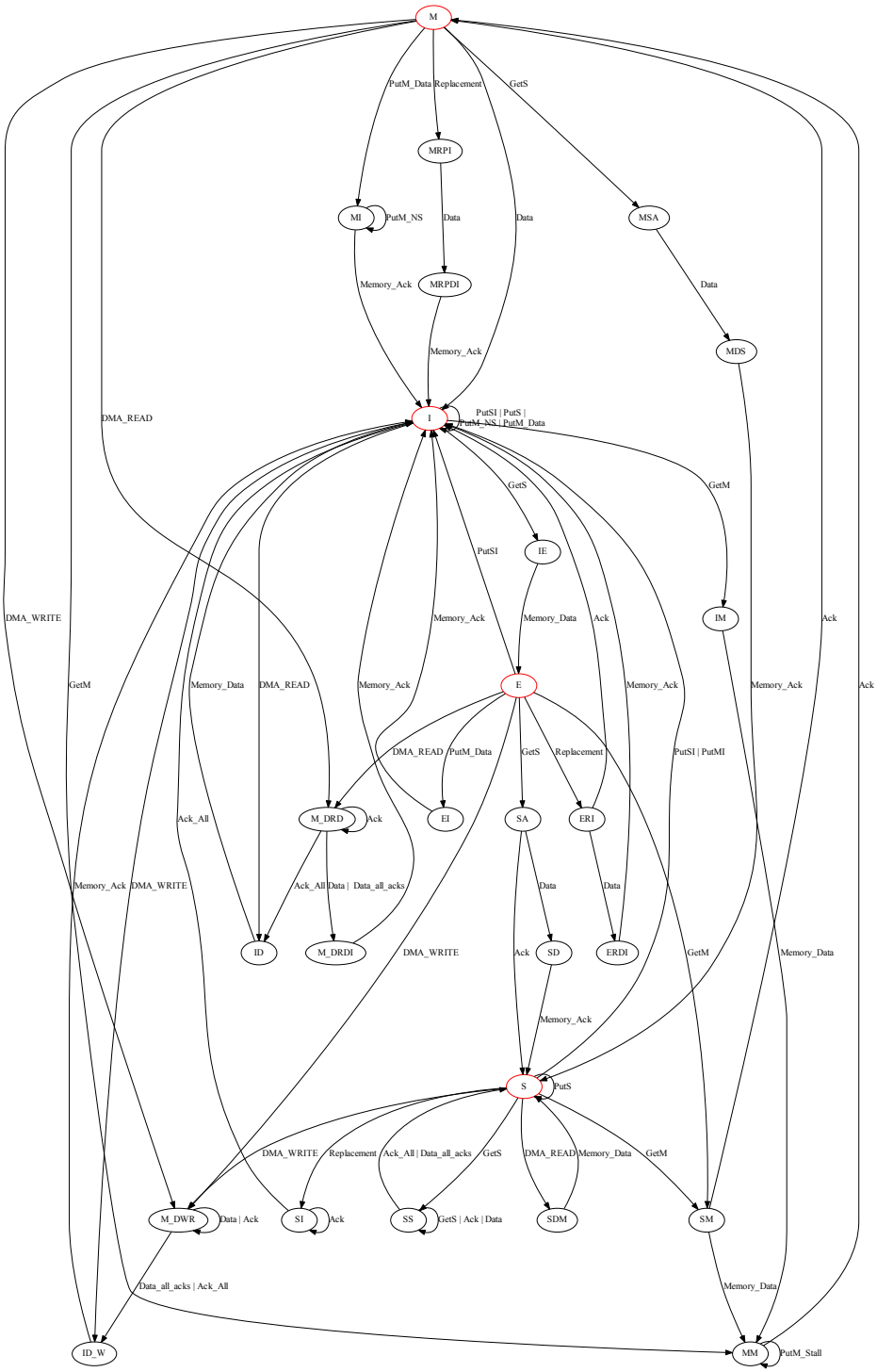
Popis událostí Home Agenta je popsán v tabulce 3.19.

Stav	Iniciátor	Popis
Data	L3 cache	Data pro zápis z L3 cache
Data_all_acks	L3 cache	Data pro zápis z L3 cache, všechna potvrzení
Memory_Data	Paměť	Data z paměti
Memory_Ack	Paměť	Potvrzení o dokončeném zápisu do paměti
DMA_READ	Řadič DMA	Žádost o čtení
DMA_WRITE	Řadič DMA	Žádost o zápis (obsahuje data)
GetS	L3 cache	Žádost o data ke čtení
GetM	L3 cache	Žádost o data pro zápis
PutS	L3 cache	Oznámení o uvolnění datového bloku z L3 cache
PutSI	L3 cache	Oznámení o uvolnění datového bloku z L3 cache, poslední sdílející
PutS_NS	L3 cache	Oznámení o uvolnění datového bloku z L3 cache, odesílající L3 není v adresáři
PutM_Data	L3 cache	Žádost o zápis dat do paměti
PutM_Stall	L3 cache	Totéž jako PutM_Data, ale je potřeba tuto žádost pozastavit
PutM_NS	L3 cache	Totéž jako PutM_Data, ale odesílatel již nemá právo k zápisu, odešle se potvrzení o zápisu, data se zahodí protože už nejsou aktuální
Ack	L3 cache	Potvrzení
Ack_All	L3 cache	Všechna potvrzení

Tabulka 3.19: Popis událostí Home Agenta

3.7.5 Home Agent – řadič

Přechodový diagram řadiče Home Agenta je znázorněn na obrázku 3.6.



Obrázek 3.6: Přechodový diagram Home Agenta, stabilní stavy jsou vyznačeny červeně

3.8 Řadič DMA

Implementace řadiče DMA byla převzata z existující implementace MESI protokolu. Pro průchod testem DMA bylo potřeba do zpráv vysílaných DMA řadičem doplnit pole „Requestor“, protože Home Agent potřebuje vědět kam má poslat odpověď.

3.8.1 DMA – Propojení

Řadič DMA je připojen k virtuálním sítím 6 a 7, které spojují DMA řadiče, Home Agenty a L3 cache. Požadavky přicházejí řadiči DMA přes frontu *mandatoryQueue*, ke které je připojen DMA sekvencer.

3.8.2 DMA – Stavy

Stavy řadiče DMA jsou popsány v tabulce 3.20.

Stav	Popis
READ	Čeká na požadavky od DMA zařízení
BUSY_RD	Vyslána žádost o čtení, čeká na data
BUSY_WRITE	Odeslána data pro zápis, čeká na potvrzení o zápisu

Tabulka 3.20: Popis stavů řadiče DMA

3.8.3 DMA – události

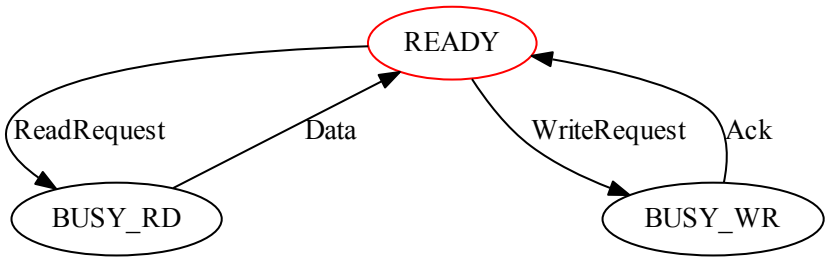
Události řadiče DMA jsou popsány v tabulce 3.21

Událost	Popis
ReadRequest	Žádost o čtení
WriteRequest	Žádost o zápis
Data	Data pro čtení
Ack	Potvrzení o dokončení zápisu.

Tabulka 3.21: Popis událostí řadiče DMA

3.8.4 DMA – řadič

Přechodový diagram řadiče DMA je na obrázku 3.7.



Obrázek 3.7: Přechodový diagram řadiče DMA, stabilní stav je vyznačen červeně

3.9 Parametry simulace

Použité parametry pro simulaci jsou použity stejné jako u procesoru Intel Xeon E5-2690 [5].

- Počet jader: 16
- Počet Home Agentů: 2
- Počet jader na procesor: 8
- Frekvence procesoru: 2.9GHz
- Velikost datového bloku v cache: 64B
- L1 cache:
  - Oddělená instrukční a datová cache
  - Stupeň asociativity: 8
  - Velikost: 32kB
- L2 cache:
  - Stupeň asociativity: 8
  - Velikost: 512kB celkem na jednu L2 cache, 256kB na segment (bank)
- L3 cache:
  - Stupeň asociativity: 20
  - Velikost: 20MB celkem, 2560kB na segment (bank)

Kapitola 4

Testování

V této kapitole je popsáno použité testování koherenčního protokolu. Testování koherenčního protokolu bylo provedeno pomocí sekvenčních a náhodných testů, které jsou popsány v sekci 4.1. Implementace byla dále ověřena pomocí vybraných benchmarků PARSEC ([6]) a SPLASH-2 ([7]). Formální ověření koherenčního protokolu nebylo provedeno.

4.1 Vestavěné testy v paměťovém systému Ruby

V paměťovém systému Ruby jsou dostupné dva testy, které pomáhají s ověřením správné funkce koherenčního protokolu.

4.1.1 Sekvenční test

Sekvenční test se snaží přistupovat do souvislého adresního prostoru. Tento test je vhodný pro základní ověření koherenčního protokolu a umožňuje rychlé odhalení závažných chyb. Tento test rovněž umožňuje testování DMA operací.

Spuštění sekvenčního testu pro 8 procesorů a 20000 operací čtení:

```
./build/X86/gem5.opt ./configs/example/ruby_mem_test.py
        --num-cpus 8 --maxloads=20000
```

Spuštění sekvenčního testu s testováním DMA operací pomocí jednoho řadiče DMA pro 8 procesorů a 20000 operací čtení:

```
./build/X86/gem5.opt ./configs/example/ruby_mem_test.py
        --num-cpus 8 --num-dmas=1 --maxloads=20000
```

4.1.2 Náhodný test

Náhodný test přistupuje k náhodným adresám z celého adresního prostoru. Tento test lépe simuluje reálné aplikace, než sekvenční test a umožňuje odhalit většinu chyb v koherenčním protokolu.

Spuštění náhodného testu pro 8 procesorů a 20000 operací čtení:

```
./build/X86/gem5.opt ./configs/example/ruby_random_test.py
        --num-cpus 8 --maxloads=20000
```

4.1.3 Parametry testů

Oba testy mají několik společných parametrů:

- Počet jader
- Počet L2 a L3 cache pamětí a počet Home Agentů
- Velikosti cache pamětí. Pro začátek je lepší začít s většími cache paměťmi, protože nám pomohou odhalit závažné chyby. Po odstranění závažných chyb je vhodné velikosti cache pamětí postupně snižovat a odstraňovat chyby spojené zejména s uvolňováním a zneplatňováním cache bloků. Výchozí nastavení velikostí cache:
  - L1 datová cache: 256B
  - L1 instrukční cache: 256B
  - L2 cache: 512B
  - L3 cache: 1kB
- Asociativita cache pamětí.
- Počet operací čtení
- Náhodné semínko (pouze pro náhodný test)
- Počet DMA zařízení (pouze sekvenční test)

Ladění koherenčního protokolu je složité a velmi zdlouhavé. Pro začátek je dobré začínat s malým systémem jednoho až dvou jader procesoru, s většími cache paměťmi s vyšším stupněm asociativity a po odstranění závažných chyb postupně zmenšovat velikosti a stupně asociativity cache pamětí a zvyšovat velikost systému.

Zde uvádím několik nejčastějších typů chyb:

- Neplatný přechod je jednou z nejčastěji se objevujících se chyb. Nastává v situaci, kdy není dostupný žádný přechod pro příchozí událost. Tato chyba je obvykle snadno řešitelná přidáním nového přechodu.
- Uváznutí (deadlock) je náročné na odhalení. Pokud dojde k uváznutí, simulace se ukončí a jediná informace, kterou simulátor vypíše, je adresa datového bloku. Pro odhalení příčiny uváznutí je potřeba pečlivě projít trasovací výstup ze simulátoru a zjistit příčinu uváznutí. Příčina uváznutí

nemusí být vždy spojena s adresou, která je vypsána simulátorem po ukončení simulace, ale s úplně jiným datovým blokem, který je například v procesu náhrady datového bloku v některé z cache paměti vyšší úrovně, která čeká na odpověď z nižší úrovně, kterou nikdy nedostane.

- Nekoherentní data - tato situace většinou nastává v případě, kdy nejsou správně obslouženy požadavky Inv a Recall. Data jsou například přítomna v jedné z L1 cache pro zápis a v druhé L1 cache ke čtení. Vestavěné testy v Ruby systému přesně vědí, jaká data mají jednotlivá jádra procesoru dostat. Pokud některé jádro procesoru dostane neplatná data, simulace se ukončí s příslušným chybovým výstupem, který mimo jiné obsahuje adresu datového bloku, očekávaná data a číslo jádra který dostal nesprávná data.

- Pády simulace nastávají velmi zřídka. Většinou je tato situace způsobena čtením dat z nesprávné sítě, přístupem k datům ve stavu Invalid nebo přístupem k nealokovanému MSHR registru.

Koherenční protokol byl testován náhodným testem pro 20 milionů operací čtení a zápisu pro několik konfigurací systému.

4.2 Modifikace simulátoru GEM5

Pro spuštění benchmarků v režimu „Syscall Emulation“ bylo potřeba upravit funkci „functionalRead“, která je použita v implementaci emulace některých systémových volání, v souboru:

`GEM5_HOME/src/mem/ruby/system/System.cc`

Nejprve bylo potřeba odstranit ověření počtu řadičů cache, které jsou ve stavu „Read_Write“, protože v naší implementaci může být více takovýchto řadičů vzhledem k inkluzivní hierarchii.

Dalším problémem, který způsoboval pády simulace, byl funkcionální přístup k datovým blokům, které jsou v pamětech cache ve stavu „Busy“, tedy v tranzientním stavu a nemusí obsahovat platná data. Tento problém je řešen pomocí funkcí, které již byly implementovány v simulátoru GEM5, jedná se o čtení zpráv přímo z bufferů řadičů cache (funkce „functionalReadBuffers“), čtení dat z bufferů síťových prvků a na závěr čtení dat bufferů řadiče hlavní paměti.

Následující část zdrojového kódu byla přidána na konec funkce „functionalRead“.

```
//try to READ data from buffers from cache controllers
for (unsigned int i = 0; i < num_controllers;++i) {
   if (m_abs_cntrl_vec[i]->functionalReadBuffers(pkt))
      return true;
}

//try to READ data from messages in network
if (m_network->functionalRead(pkt)){
   return true;
}

//try to READ data from memory
for (unsigned int i = 0;
        i < m_memory_controller_vec.size() ;++i) {
  if (m_memory_controller_vec[i]->functionalReadBuffers(pkt))
     return true;
}

return false;
```

4.3 Sada benchmarků PARSEC

PARSEC benchmark (Princeton Application Repository for Shared-Memory Computers [6]) je sada benchmarků pro porovnání výkonností procesorů. Tyto benchmarky používají několik různých metod paralelizace:

- pthreads [8]
- Intel TBB (Threading Building Blocks) [9]
- OpenMP [10]

Vzhledem ke skutečnosti, že máme k dispozici pouze knihovnu m5threads (upravená knihovna pthreads pro simulátor GEM5), se omezíme pouze na benchmarky, které jsou založené na pthread knihovně.

Každý benchmark ze sady PARSEC obsahuje šest velikostí vstupů:

- Test – velmi malý vstup pro otestování funkčnosti programu,
- Dev – velmi malý vstup určený pro testování a vývoj,
- Small, Medium a Large – vstupy pro vývoj architektur pomocí simulátorů,
- Native – velký vstup pro porovnání reálných systémů.

Je velice těžké vybrat rozumný vstup pro porovnání reálného a simulovaného systému. Například u reálného systému je doba benchmarku velmi nízká, řádově desítky milisekund. Simulace se stejným vstupem může trvat jednotky ale i desítky hodin. U většiny benchmarků nejlépe vyhovoval vstup *Small* s reálnou dobou běhu přibližně 40 - 300ms a dobou simulace do 10 hodin pro jedno vlákno.

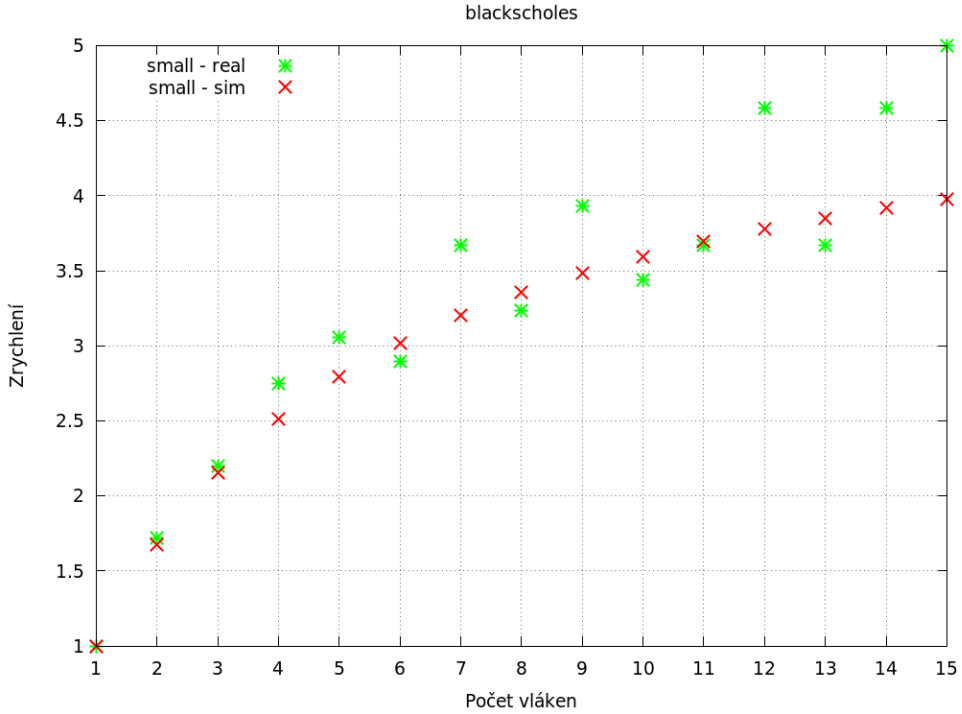
Spuštění benchmarků v režimu *syscall emulation* je problematické. Prvním problémem je kompilace benchmarku pro simulátor, program musí být staticky slinkovaný a knihovna *pthreads* musí byt nahrazena zjednodušenou knihovnou *m5threads*. Některé knihovny musí být upraveny pro běh pod simulátorem. Dalším problémem jsou chybějící implementace velkého množství systémových volání, potřebných pro běh benchmarků například: *socket, mprotect, madvise, openat atd.* Po úspěšné kompilaci a spuštění simulace se může stát, že se daná simulace ukončí s chybovou zprávou týkající se špatného přístupu do paměti.

Simulace probíhala s parametry uvedenými v sekci 3.9. Měření probíhalo s 1 – 15 vlákny nebo s 1, 2, 4 a 8 vlákny (benchmarky, které vyžadují počet vláken jako mocninu dvou), protože knihovna *m5threads* vyžaduje jedno vlákno navíc, které slouží jako řídící.

4.3.1 Blackscholes

Benchmark blackscholes počítá ceny portfolia pomocí Black-Scholes PDE (parciální diferenciální rovnice). Tento benchmark se vyznačuje nízkou úrovní sdílení dat a jejich výměny mezi jednotlivými výpočetními vlákny.

Na obrázku 4.1 vidíme výsledky s malým vstupem. Simulace MOESI protokolu téměř kopíruje zrychlení dosažené na fyzickém stroji, ale s rostoucím počtem vláken trochu ztrácí.

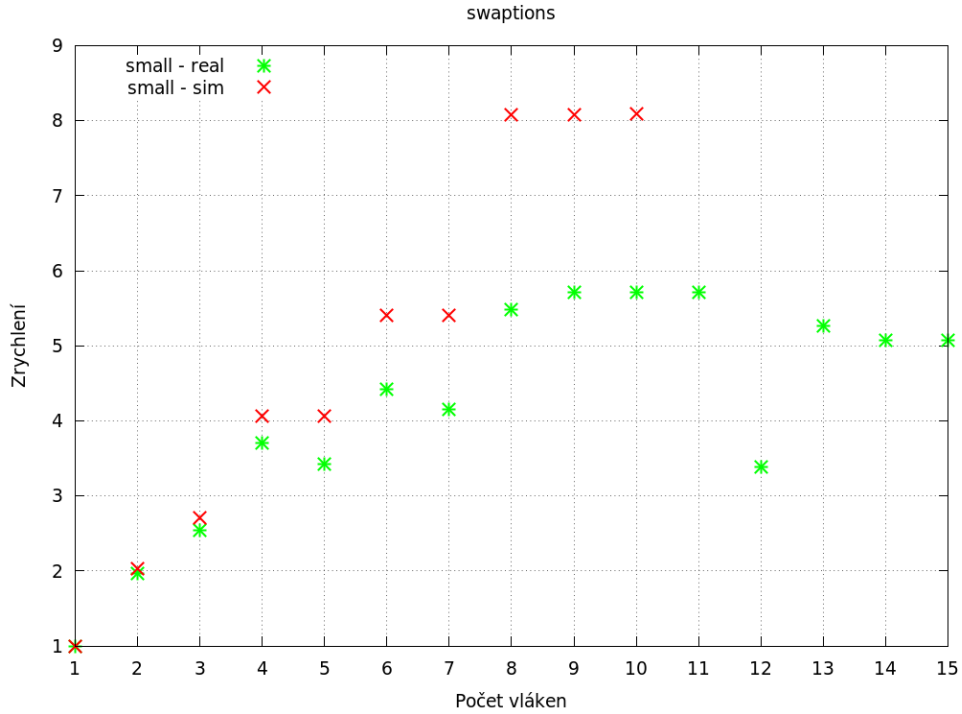


Obrázek 4.1: Blackscholes benchmark – paralelní zrychlení

4.3.2 Swaptions

Benchmark swaptions počítá ceny portfolia pomocí simulace Monte Carlo. Sdílení dat a výměna dat mezi výpočetními vlákny je používáno málo.

Na obrázku 4.2 jsou výsledky měření pro malý vstup. Simulace MOESI protokolu škáluje lépe než běh na reálném systému.



Obrázek 4.2: Swaptions benchmark – paralelní zrychlení

Simulace pro 11 a 12 vláken nedoběhla z důvodu neimplementovaného systémového volaní *socket*. Simulace s 13 vlákny skončila kvůli špatnému přístupu benchmarku do paměti. Simualce pro 14 a 15 vláken nedoběhla ani po 48 hodinách, pro ostatní počet vláken simulace doběhla do 11 hodin.

4.4 Sada benchmarků SPLASH-2

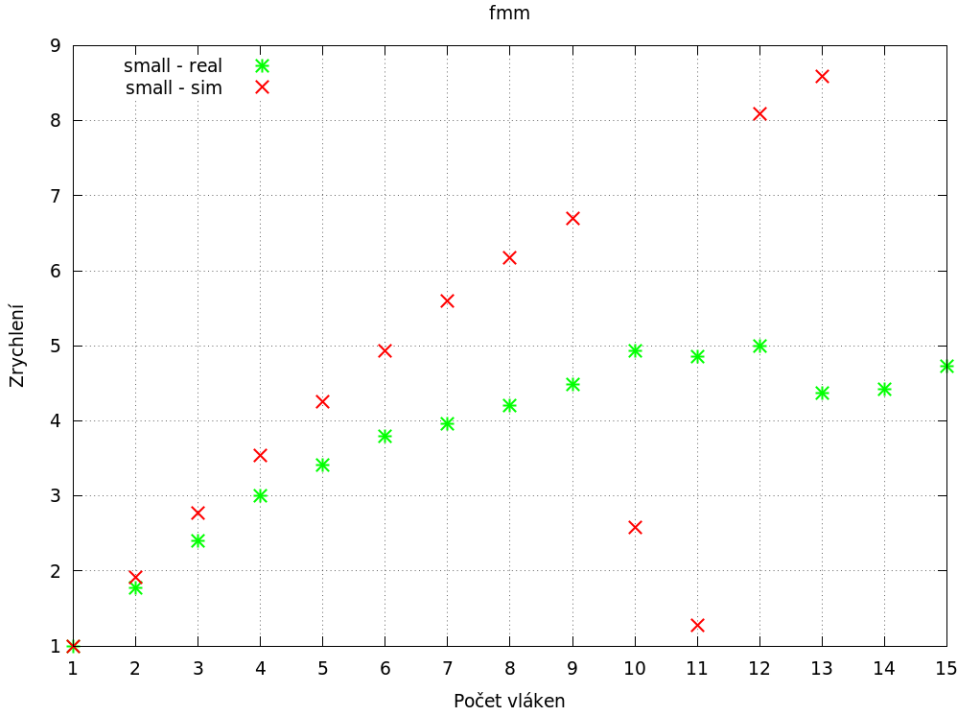
SPLASH-2 benchmark (Stanford Parallel Applications for Shared Memory [7]) je sada aplikací pro srovnání výkonu víceprocesorových systémů.

Vstupy pro benchmarky ze sady SPLASH-2 jsou rozděleny podobně jako u sady PARSEC s tím rozdílem, že u většiny benchmarků se mění vstupní parametry benchmarku, ale nemění se vstupní soubory.

Situace se spuštěním benchmarků ze sady SPLASH-2 v režimu *syscall emulation* je velmi podobná jako s těmi ze sady PARSEC.

4.4.1 FMM

Benchmark FMM simuluje interakce v systému objektů v 2D prostoru.



Obrázek 4.3: FMM benchmark – paralelní zrychlení

Na obrázku 4.3 jsou výsledky měření pro malý vstup. Simulace MOESI protokolu škáluje lépe, než běh na reálném systému, s výjimkou velkého propadu simulace s 10 a 11 vlákny. Tento propad je způsoben velikým rozdílem počtu operací *Store* ve stavu S a ve stavu M viz tabulka 4.1, celkový počet operací *Load* stoupl v porovnání se simulacemi s 9 a 12 vlákny. Kompletní statistiky simulací jsou na přiloženém CD.

Simulace pro 14 a 15 vláken nedoběhla ani po 60 hodinách, ostatní simulace doběhly do 10 hodin.

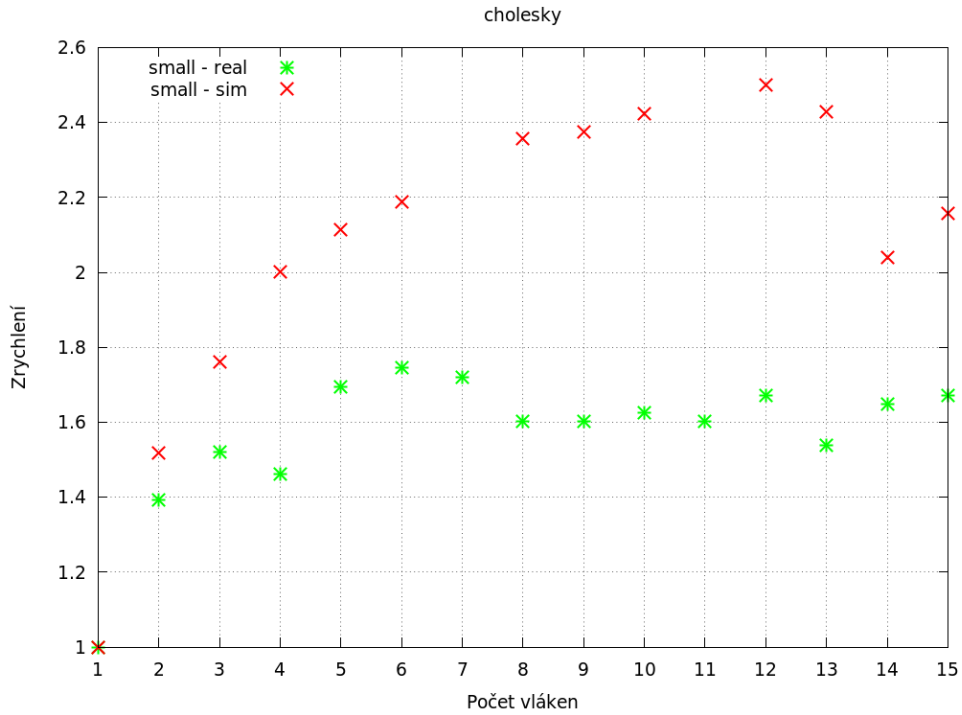
Počet vláken	Stav a operace			
	S.Load	S.Store	M.Load	M.Store
9	242526949	628949	188395613	128030757
10	308084801	4605259	198500129	88465837
11	529653238	11998792	210023111	98116847
12	287774432	597489	188514007	172925590

Tabulka 4.1: FMM – rozdíly v počtu operací

4.4.2 Cholesky

Benchmark Cholesky provádí Cholského dekompozici čtvercové matice na součin dolní a horní trojúhelníkové matice.

Na obrázku 4.4 jsou výsledky měření pro malý vstup. Simulace škáluje lépe než běh na reálném systému. To je způsobeno především hlavní vlastností MOESI protokolu, tedy sdílením modifikovaného datového bloku bez zápisu do hlavní paměti.



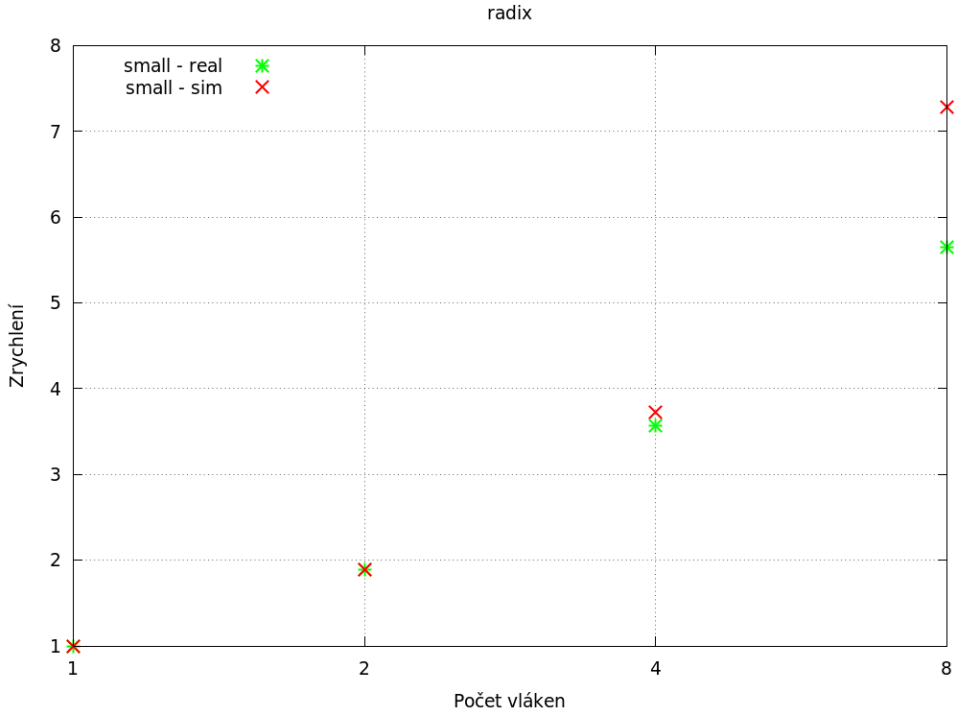
Obrázek 4.4: Cholesky benchmark – paralelní zrychlení

Simulace pro 7 a 11 vláken běžela 67 hodin bez výsledku, ostatní doběhly do 6 hodin a 40 minut.

4.4.3 Radix

Benchmark Radix provádí celočíselné řazení pomocí řadicího algoritmu Radix sort. Tento benchmark pro svůj běh vyžaduje, aby počet vláken byla mocnina dvou.

Na obrázku 4.5 jsou výsledky měření pro malý vstup. Simulace škáluje o trochu lépe pro větší počet vláken.



Obrázek 4.5: Radix benchmark – paralelní zrychlení

Závěr

Vytvořili jsme tříúrovňovou inkluzivní hierarchii sdílených pamětí cache s integrovanou adresářovou strukturou a s koherenčním protokolem MOESI. Dále jsme upravili „funkcionální čtení“, které je používáno v implementaci některých systémových volání. Po dokončení implementace a základního testování jsme testovali funkcionalitu a škálovatelnost implementace pomocí benchmarků ze sad PARSEC a SPLASH-2 na instrukční sadě *x86* v režimu *syscall emulation*.

Implementace škáluje velmi pěkně a v některých případech dokonce lépe než na reálném systému s MESIF protokolem. Lepší škálování je především dáno vlastností MOESI protokolu, kvůli které není nutné zapisovat modifikovaná data do paměti při přechodu do sdíleného stavu. K lepšímu škálování také pomáhá sdílená L2 cache, která je sdílena dvěma L1 cache. Tím se urychlila výměna dat mezi L1 cache, které jsou připojené k jedné L2 cache. Dalším faktorem přispívajícím k lepším výsledkům, než na reálném stroji je jednoduchá simulace propojovací sítě mezi procesory. Zde není simulována vyšší latence komunikace mezi procesory. Pro lepší simulaci propojovací sítě je potřeba použít jiný model propojovací sítě nebo definovat vlastní topologii a nastavit zpoždění jednotlivým portům aktivních prvků propojovací sítě.

Myslím si, že funkční implementace MOESI protokolu se dá považovat za úspěch. Implementace by měla podporovat systémy s větším počtem procesorů a Home Agentů, ale tato možnost nebyla z časových důvodů řádně otestována.

Budoucí práce

- Upravit propojení více procesorů, tak, aby simulace byla co nejpřesnější.
- Doplnit do simulátoru GEM5 implementaci emulace systémových volání, potřebných pro běh benchmarků ze sad SPLASH-2 a PARSEC. Nalézt a opravit chyby v benchmarcích a použitých knihovnách, které způsobují neoprávněný přístup do paměti.
- Hlouběji analyzovat statistiky, které generuje paměťový systém *Ruby* a podle výsledků analýzy provést další optimalizace a jejich následné otestování.

Literatura

[1] Binkert, N.; Beckmann, B.; Black, G.; aj.: The GEM5 Simulator. [online], cit. 7. 4. 2015. Dostupné z: http://dl.acm.org/citation.cfm?doid=2024716.2024718

[2] Coherence-Protocol-Independent Memory Components. [online], cit. 7. 4. 2015. Dostupné z: http://www.m5sim.org/Coherence-Protocol-Independent_Memory_Components

[3] GEM5 Documentation. [online], cit. 7. 4. 2015. Dostupné z: http://www.gem5.org/Documentation

[4] Sorin, D. J.; Hill, M. D.; Wood, D. A.: *A Primer on Memory Consistency and Cache Coherence*. cit. 7. 4. 2015. Dostupné z: https://lagunita.stanford.edu/c4x/Engineering/CS316/asset/A_Primer_on_Memory_Consistency_and_Coherence.pdf

[5] Intel® Xeon® Processor E5-1600/E5-2600/E5-4600 Product Families. [online], cit. 27. 4. 2015. Dostupné z: http://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-1600-2600-vol-1-datasheet.html

[6] PARSEC – Oficiální stránka. [online], cit. 7. 4. 2015. Dostupné z: http://parsec.cs.princeton.edu/index.htm

[7] Woo, S. C.; Ohara, M.; Torrie, E.; aj.: The SPLASH-2 programs: characterization and methodological considerations. [online], cit. 7. 4. 2015. Dostupné z: http://dl.acm.org/citation.cfm?doid=225830.223990

[8] POSIX Threads Programming. [online], cit. 27. 4. 2015. Dostupné z: https://computing.llnl.gov/tutorials/pthreads/

[9] Intel® Threading Building Blocks. [online], cit. 27. 4. 2015. Dostupné z: https://software.intel.com/en-us/intel-tbb

[10] The OpenMP® API. [online], cit. 27. 4. 2015. Dostupné z: http://openmp.org/wp/

Příloha **A**

Seznam použitých zkratek

SLICC Specification Language for Implementing Cache Coherence

MSHR Miss Status Handling Registers

TBE Transaction Buffer Entry

PARSEC Princeton Application Repository for Shared-Memory Computers

SPLASH-2 Stanford Parallel Applications for Shared Memory

DMA Direct Memory Access

LRU Least Recently Used

DDR Double Data Rate

LLC Last Level Cache

HA Home Agent

FFT Fast Fourier Transform

Příloha B

Přechodové tabulky

V této příloze jsou popsány použité akce a přechodové tabulky všech řadičů cache, Home Agenta a řadiče DMA. Každá akce má svoji zkratku, která je použita v přechodových tabulkách. Větší tabulky jsou rozděleny na dvě části. Kompletní přechodové tabulky v HTML jsou na přiloženém CD.

B.1 L1 cache

Akce	Popis
a_issueGETS	Vyšle žádost GetS do L2 cache
pa_issuePfGETS	Vyšle žádost GetS (od prefetcheru) do L2 cache
b_issueGETM	Vyšle žádost GetM do L2 cache
pb_issuePfGETM	Vyšle žádost GetM (od prefetcheru) do L2 cache
d_sendDataToRequestor	Pošle data žadateli
d2_sendDataToL2	Pošle data do L2 cache
e_sendAckToRequestor	Pošle potvrzení žadateli
eo_sendAckToOriginalRequestor	Pošle potvrzení originálnímu žadateli
g_issuePutM	Vyšle žádost PutM do L2 cache
js_sendUnblock	Pošle odblokování do L2 cache
h_load_hit	Pošle data pro čtení jádru procesoru
hh_store_hit	Pošle data pro zápis jádru procesoru
i_allocateTBE	Alokuje TBE záznam
k_popMandatoryQueue	Odebere zprávu z fronty žádostí od jádra
l_popRequestQueue	Odebere zprávu z fronty žádostí od L1/L2 cache
o_popIncomingResponseQueue	Odebere zprávu z fronty odpovědí od L1/L2 cache
s_deallocateTBE	Dealokuje TBE záznam

Tabulka B.1: Seznam akcí L1 cache

u_writeDataToL1Cache	Zapíše data do cache
q_updateAckCount	Aktualizuje počet očekávaných potvrzení
ff_deallocateL1CacheBlock	Dealokuje L1 cache blok
oo_allocateL1DCacheBlock	Alokuje datový cache blok
pp_allocateL1ICacheBlock	Alokuje instrukční cache blok
z_stallAndWaitMandatoryQueue	Pozastaví frontu žádostí od jádra
zo_stallAndWaitOptionalQueue	Pozastaví frontu žádostí od prefetcheru
kd_wakeUpDependents	Probudí buffery
kda_wakeUpAllDependents	Probudí všechny buffery
po_observeMiss	Informuje prefetcher o chybějícím datovém bloku
ppm_observePfMiss	Informuje prefetcher o započatém požadavku
pq_popPrefetchQueue	Odebere zprávu z fronty prefetcheru
mp_markPrefetched	Označí přednačtený blok
ps_issuePUTS	Vyšle žádost PutS do L2 cache
sa_setAckCount	Nastaví očekávaný počet potvrzení podle přijatého adresáře
dou_sendDataUnblockToOriginalRequestor	Odešle data a odblokování původnímu žadateli (L1 cache)
do_sendDataToOriginalRequestor	Odešle data původnímu žadateli (L1 cache)
zz_stallAndWaitRequestQueue	Pozastaví frontu žádostí od L2 cache
al2_sendAck	Pošle potvrzení do L2 cache
hh_flush_hit	Informuje jádro o dokončené operaci Flush

Tabulka B.2: Pokračování seznamu akcí L1 cache

Obrázek B.1: Přechodová tabulka L1 cache

B.2 L2 cache

Akce	Popis
a_issueGETS	Vyšle žádost GetS do L2 cache
d_sendDataToRequestor	Pošle data žadateli (L1 cache)
df_sendDataToRequestor	Pošle data žadateli (L2 cache)
de_sendExclusiveDataToRequestor	Pošle exkluzivní data žadateli
e_sendDataToGetSRequestors	Pošle data všem žadatelům zaznamenaným v TBE
ex_sendExclusiveDataToGetSRequestors	Pošle exkluzivní data žadateli
ee_sendDataToGetXRequestor	Pošle data žadateli uvedenému v TBE (L1 cache)
ef_sendDataToGetXRequestor	Pošle data žadateli uvedenému v TBE (L2 cache)
f_sendInvToSharers	Pošle invalidaci sdílejícím L1 cache
i_allocateTBE	Alokuje TBE záznam
s_deallocateTBE	Dealokuje TBE záznam
jj_popL1RequestQueue	Odebere zprávu z fronty žádostí od L1 cache
j3_popL3RequestQueue	Odebere zprávu z fronty žádostí od L3 cache
k_popUnblockQueue	Odebere zprávu z fronty odblokování od L1 cache
o_popIncomingResponseQueue	Odebere zprávu z fronty odpovědí od L1 cache
o3_popIncomingResponseQueue	Odebere zprávu z fronty odpovědí od L2/L3 cache
m_writeDataToCache	Zapíše data od L1 cache do cache
m3_writeDataToCache	Zapíše data z L2/L3 cache do cache
mr_writeDataToCacheFromRequest	Zapíše data z žádosti od L1 cache do cache
q_updateAck	Aktualizuje počet očekávaných potvrzení (od L1 cache)
q3_updateAck	Aktualizuje počet očekávaných potvrzení (od L2/L3 cache)
qs_SubAck	Odečte jedno potvrzení od očekávaných potvrzení

Tabulka B.3: Seznam akcí L2 cache

Akce	Popis
qq_allocateL2CacheBlock	Alokuje cache blok
rr_deallocateL2CacheBlock	Dealokuje cache blok
t_sendWBAck	Pošle potvrzení o zápisu
zz_stallAndWaitL1RequestQueue	Pozastaví frontu požadavků od L1 cache
zf_stallAndWaitL3RequestQueue	Pozastaví frontu požadavků od L3 cache
zn_recycleResponseNetwork	Recykluje frontu odpovědí od L3 cache
kd_wakeUpDependents	Probudí buffery
aw_issueGETM	Vyšle žádost GetM do L3 cache
ac_sendAck	Pošle potvrzení do L1 cache
ai_sendAck	Pošle potvrzení do L3 cache podle TBE
a3_sendAckToL3	Pošle potvrzení do L3 cache
do_sendDataToOriginalRequestor	Pošle data původnímu žadateli (L2 cache)
eo_sendAckToOriginalRequestor	Pošle potvrzení původnímu žadateli (L2 cache)
sr_sendRecallToSharers	Pošle zneplatnění sdílejícím L1 cache
sd_sendDowngrade	Pošle Downgrade L1 cache
ps_issuePUTS	Pošle žádost PutS do L3 cache
dd_sendDataDir	Pošle data a adresář L1 cache
dus_sendDataUnblock	Pošle data a odblokování jiné L2 cache
g_issuePutM	Pošle žádost PutM
sfs_sendFwdGETS	Přepošle žádost Fwd_GetS
sf_sendFwdGETS	Vyšle žádsot Fwd_GetS
sfm_sendFwdGETM	Vyšle žádost Fwd_GetM
s3_sendDataToL3	Pošle data do L3 cache
sa_setAckCount	Nastaví počet očekávaných potvrzení podle příchozího adresáře
sac_setAckCount	Nastaví počet očekávaných potvrzení podle lokálního adresáře
ddm_sendDataDirToGetMRequestor	Pošle data a adresář do L1 cache
su3_sendUnblockToL3	Pošle odblokování L3 cache

Tabulka B.4: Pokračování seznamu akcí L2 cache

Obrázek B.2: Přechodová tabulka L2 cache

Obrázek B.3: Přechodová tabulka L2 cache – pokračování

B.3 L3 cache

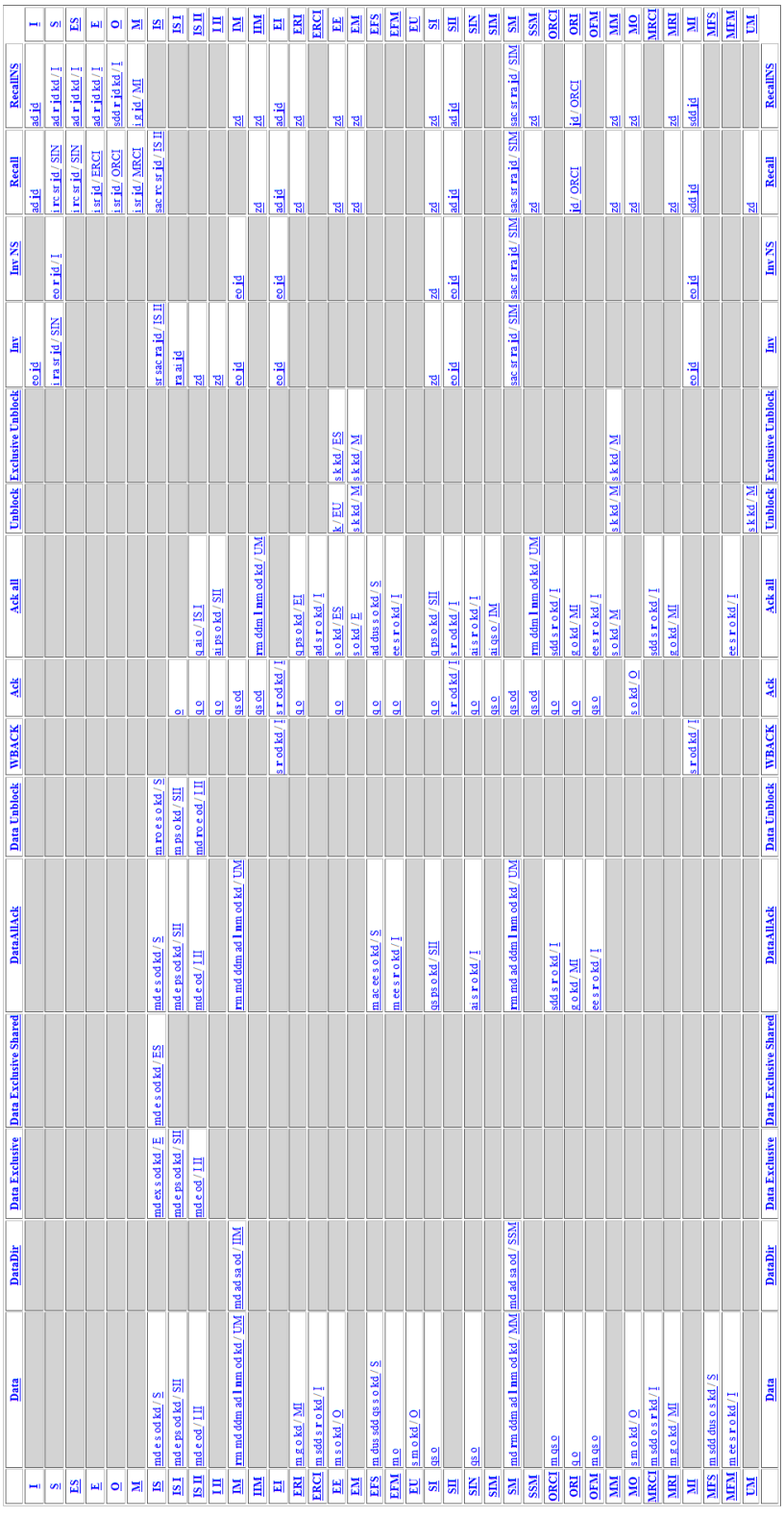
Akce	Popis
a_issueGETS	Vyšle žádost GetS do HA
d_sendDataToRequestor	Pošle data žadateli (L2 cache)
df_sendDataToRequestor	Pošle data žadateli (L3 cache)
de_sendExclusiveDataToRequestor	Pošle exkluzivní data žadateli
e_sendDataToGetSRequestors	Pošle data všem žadatelům zaznamenaným v TBE
ex_sendExclusiveDataToGetSRequestors	Pošle exkluzivní data žadateli
ee_sendDataToGetXRequestor	Pošle data žadateli uvedenému v TBE (L2 cache)
f_sendInvToSharers	Pošle invalidaci sdílejícím L1 cache
i_allocateTBE	Alokuje TBE záznam
s_deallocateTBE	Dealokuje TBE záznam
jj_popL2RequestQueue	Odebere zprávu z fronty žádostí od L2 cache
jd_popDirRequestQueue	Odebere zprávu z fronty žádostí od HA
k_popUnblockQueue	Odebere zprávu z fronty odblokování od L2 cache
o_popIncomingResponseQueue	Odebere zprávu z fronty odpovědí od L2 cache
od_popIncomingResponseQueue	Odebere zprávu z fronty odpovědí od HA/L3 cache
m_writeDataToCache	Zapíše data od L2 cache do cache
md_writeDataToCache	Zapíše data od HA do cache
mr_writeDataToCacheFromRequest	Zapíše data z žádosti od L2 cache do cache
q_updateAck	Aktualizuje počet očekávaných potvrzení (od L2 cache)
qs_SubAck	Odečte jedno potvrzení od očekávaných potvrzení
qq_allocateL2CacheBlock	Alokuje cache blok
rr_deallocateL2CacheBlock	Dealokuje cache blok

Tabulka B.5: Seznam akcí L3 cache

Akce	Popis
t_sendWBAck	Pošle potvrzení o zápisu
zz_stallAndWaitL2RequestQueue	Pozastaví frontu požadavků od L1 cache
zd_stallAndWaitDirRequestQueue	Pozastaví frontu požadavků od HA
kd_wakeUpDependents	Probudí buffery
aw_issueGETM	Vyšle žádost GetM do HA
ac_sendAck	Pošle potvrzení do L2 cache
ai_sendAck	Pošle potvrzení do HA podle TBE
ad_sendAckToDir	Pošle potvrzení do HA
do_sendDataToOriginalRequestor	Pošle data původnímu žadateli (L3 cache)
eo_sendAckToOriginalRequestor	Pošle potvrzení původnímu žadateli (L3 cache)
sr_sendRecallToSharers	Pošle zneplatnění sdílejícím L2 cache
sd_sendDowngrade	Pošle Downgrade do L2 cache
ps_issuePUTS	Pošle žádost PutS do HA
dd_sendDataDir	Pošle data a adresář do L2 cache
dus_sendDataUnblock	Pošle data a odblokování jiné L3 cache
g_issuePutM	Pošle žádost PutM
sfs_sendFwdGETS	Vyšle žádost Fwd_GetS
sfm_sendFwdGETM	Vyšle žádost Fwd_GetM
sdd_sendDataToDir	Pošle data do L3 cache
sa_setAckCount	Nastaví počet očekávaných potvrzení podle příchozího adresáře
sac_setAckCount	Nastaví počet očekávaných potvrzení podle lokálního adresáře
ddm_sendDataDirToGetMRequestor	Pošle data a adresář do L2 cache

Tabulka B.6: Pokračování seznamu akcí L3 cache

Obrázek B.4: Přechodová tabulka L3 cache



Obrázek B.5: Přechodová tabulka L3 cache –pokračování

B.4 Home Agent

Akce	Popis
a_sendAck	Pošle potvrzení žadateli
d_sendDataExclusive	Pošle exkluzivní data
j_popIncomingRequestQueue	Odebere zprávu z fronty požadavků
k_popIncomingResponseQueue	Odebere zprávu z fronty odpovědí
l_popMemQueue	Odebere zprávu z fronty odpovědí od hlavní paměti
kd_wakeUpDependents	Probudí buffery
qf_queueMemoryFetchRequest	Pošle požadavek na čtení z hlavní paměti
qw_queueMemoryWBRequest	Pošle požadavek na zápis do hlavní paměti
m_writeDataToMemory	Zapíše data do paměti adresáře
mr_writeDataToMemory	Zapíše data z požadavku do paměti adresáře
qf_queueMemoryFetchRequestDMA	Pošle požadavek na čtení do hlavní paměti (z DMA)
dr_sendDMAData	Pošle data řadiči DMA
dw_writeDMAData	Zapíše data od DMA řadiče do paměti adresáře
qw_queueMemoryWBRequest_partial	Pošle požadavek na zápis do hlavní paměti (pouze částečný zápis)
z_stallAndWaitRequest	Pozastaví frontu požadavků
inv_sendCacheInvalidate	Pošle invalidaci sdílejícím L3 cache
drp_sendDMAData	Pošle data řadiči DMA
v_allocateTBE	Alokuje TBE záznam
dwt_writeDMADataFromTBE	Zapíše data do paměti adresáře z TBE záznamu
qw_queueMemoryWBRequest_partialTBE	Pošle požadavek na částečný zápis do hlavní paměti (pouze částečný zápis)
w_deallocateTBE	Dealokuje TBE záznam

Tabulka B.7: Seznamu akcí Home Agenta

Akce	Popis
as_AddSharer	Přidá žadatele do adresáře (po odpovědi od hlavní paměti)
aas_AddSharer	Přidá žadatele do adresáře
cs_clearSharers	Vymaže adresář
sr_sendRecall	Pošle zneplatnění sdílejícím L3 cache
sfs_sendFwdGetS	Vyšle požadavek Fwd_GetS
rs_removeSharer	Odebere žadatele z adresáře (fronta žádostí)
rss_removeSharer	Odebere žadatele z adresáře (fronta odpovědí)
sfm_sendFwdGetM	Vyšle požadavek Fwd_GetM
dd_sendDataDir	Pošle data a adresář
q_updateAck	Aktualizuje počet očekávaných potvrzení
qs_subAck	Odečte jedno potvrzení z očekávaných potvrzení
qa_addAck	Přičte jedno potvrzení k očekávaným potvrzením
sa_setAckCount	Nastaví počet očekávaných potvrzení podle adresáře
aaw_sendWBAck	Pošle žadateli potvrzení o zápisu
qww_queueMemoryWBRequest	Pošle požadavek na zápis do hlavní paměti
so_setOwner	Nastaví vlastníka datového bloku
rd_recordDMARequestor	Zapíše identifikátor DMA zařízení do TBE záznamu
sda_sendAckToDMARequestor	Pošle potvrzení řadiči DMA

Tabulka B.8: Pokračování seznamu akcí Home Agenta

Obrázek B.6: Home Agent – Přechodová tabulka

B.5 DMA řadič

Akce	Popis
s_sendReadRequest	Vyšle požadavek DMA_READ Home Agentu
s_sendWriteRequest	Vyšle požadavek DMA_WRITE Home Agentu
a_ackCallback	Informuje DMA sekvencer o dokončeném zápisu
d_dataCallback	Pošle data DMA sekvenceru
p_popRequestQueue	Odebere zprávu z fronty žádostí od sekvenceru
p_popResponseQueue	Odebere zprávu z fronty odpovědí

Tabulka B.9: Seznam akcí řadiče DMA

	ReadRequest	WriteRequest	Data	Ack	
READY	s p / BUSY RD	s p / BUSY WR			READY
BUSY RD			d p / READY		BUSY RD
BUSY WR				a p / READY	BUSY WR
	ReadRequest	WriteRequest	Data	Ack	

Obrázek B.7: Přechodová tabulka řadiče DMA

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `html` ..... přechodové tabulky v HTML
- `src`
  - `impl` ..... zdrojové kódy implementace a konfigurační soubory
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Capek_Jindrich_2015.pdf` ..... text práce ve formátu PDF
- `gem5-stable` ..... zdrojové kódy GEM5 s MOESI protokolem
- `diagrams` ..... přechodové diagramy řadičů cache
- `out` ..... výstupy benchmarků a statistiky Ruby systému