Elektrotehnički fakultet
Univerziteta u Beogradu
Katedra za računarsku tehniku i informatiku

Kraći pregled i Vivio simulacije *snoopy* protokola koherencije keš memorija

- prateća dokumentacija -

Verzija: 1.2.2009. 20:08

Sadržaj

1. UVOD 3

2. PONIŠTAVAJUĆI PROTOKOLI 3

2.1. WTI 3
2.2. MSI 3
2.3. MESI 4
2.4. MOESI 5

3. AŽURIRAJUĆI PROTOKOLI 6

3.1. *DRAGON* 6
3.2. *FIREFLY* 8

4. VIVIO SIMULACIJE 9

4.1. MODEL, KLJUČNE KLASE I METODE 9

5. PRAVCI DALJEG RAZVOJA 10

6. ZAKLJUČAK 10

7. REFERENCE 10

8. SPISAK IZMENA NA DOKUMENTU 11

1. Uvod

Protokoli za koherenciju keš memorije (engl. *cache coherence*) protokoli se koriste u multiprocesorskim sistemima kako bi se očuvala atomičnost i redosled upisa i čitanja podataka od strane više procesora sa zasebnim keš memorijama. Jedna važna grupa ovih protokola su oni kod kojih procesori imaju dodatnu logiku za praćenje saobraćaja na magistrali (engl. *snoopy protocols*). Radi jednostavnosti, protokoli ove grupe su kategorisani prema načinu rada u 2 vrste, poništavajuće i ažurirajuće. Poništavajući (engl. *invalidate*) protokoli prilikom upisa u deljenu keš liniju (keš liniju čiji se sadržaj nalazi i u keš memorijama drugih procesora) vrše poništavanje (engl. *invalidation*) tih ulaza u drugim procesorima. Ažurirajući (engl. *update*) protokoli pri upisu podataka u neku deljenu keš liniju vrše ažuriranje sadržaja kod ostalih keš memorija i, u zavisnosti od protokola, u operativnoj memoriji.

Vivio [1] je biblioteka za prikaz animacija koja u okviru sebe definiše programski jezik koji ima za cilj da upravlja tim animacijama. Animacije se najčešće prikazuju u okviru *web* pretraživača, gde je omogućena interakcija korisnika sa njima. Zbog postojećih animacija koje prikazuju uprošćene simulacije *cache-coherence* protokola [1.1], [1.2], kao i činjenice da se kao edukativni alat pokazao korisnim, Vivio je odabran kao platforma za pravljenje animacija za preostale protokole. Gde god je to bilo moguće, upotrebljen je programski kod postojećih animacija.

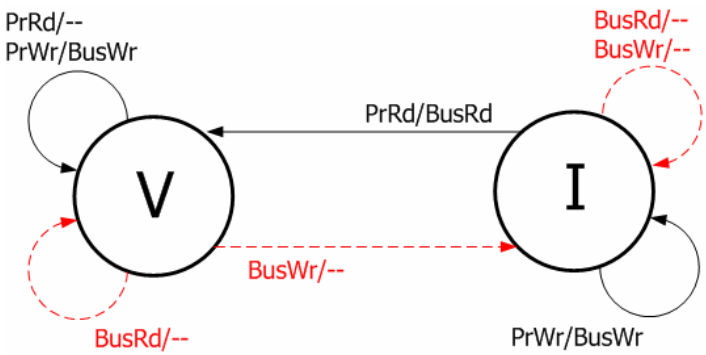
Sve animacije sa uprošćenim simulacijama uzimaju u obzir situaciju sa 4 memorijske adrese u operativnoj memoriji i 4 procesora gde svaki poseduje keš memoriju veličine 2 linije. Svaka animacija prikazuje sadržaj i stanja svih elemenata. Svaki od procesora može da vrši instrukcije čitanja i upisa. Svaku od instrukcija zadaje korisnik. Reakcija animacije na instrukciju korisnika je uvek ispraćena uočljivim pokazateljima.

2. Poništavajući protokoli

Objašnjenja protokola u ovom i sledećem odeljku su preuzeta iz [2] i [3].

2.1. WTI

WTI je skraćenica od *Write-Through Invalidate*. Ovaj protokol je najjednostavniji i najmanje efikasan protokol koji je u ovom dokumentu opisan. Svaki upis uvek prolazi do memorije i svaki put kada keš detektuje upis u keširanu adresu od strane drugog keša vrši se poništavanje te keš linije. Prilikom promašaja pri upisu, postoje dve mogućnosti. Kod *write-no-allocate* strategije, novi blok se ne dovlači u keš memoriju već se upis vrši direktno u operativnu memoriju, zaobilazeći keš. U slučaju *write-allocate* strategije, blok će prvo biti dovučen u keš memoriju, pa će zatim biti izvršen upis i u keš memoriju i u operativnu memoriju.

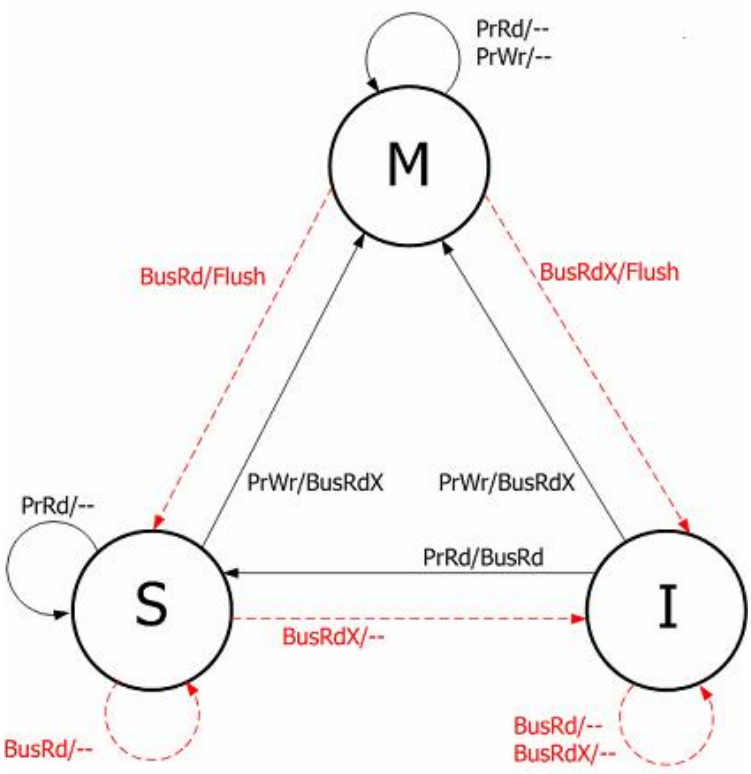


Slika 1 - WTI dijagram stanja (*write-no-allocate* strategija)

2.2. MSI

MSI protokol je dobio svoj naziv po mogućim stanjima u kojima se može naći keš linija (*Modified*, *Shared* i *Invalid*). Nastao je evolucijom WTI protokola kako bi oslobodio magistralu smanjivanjem broja poništavanja pri upisu. U ovom protokolu, u vrši se odloženi upis podatka u

memoriju. Kao posledicu, ovaj protokol donosi novinu da sadržaj podatka u operativnoj memoriji ne mora biti ažuran. Sistem prelazi u *Shared* stanje prilikom čitanja, a u *Modified* stanje prilikom upisa. Ukoliko neki keš vrši operaciju upisa u podatak koji drugi procesor ima u stanju *Modified* ili *Shared*, drugi keš je dužan da poništi svoju kopiju.Čitanje iz keš linije ne uzrokuje promenu ukoliko se podatak već nalazi u kešu. . U stanju *Shared* memorija je ažurna. Prilikom čitanja ili upisa, ukoliko se podatak nalazi u nekom kešu u stanju *Modified*, taj keš je dužan da dostavi podatak kešu koji ga zahteva i pritom ažurira memoriju. Prilikom zamene *Modified* bloka potrebno je izvršiti ažuriranje bloka u memoriji



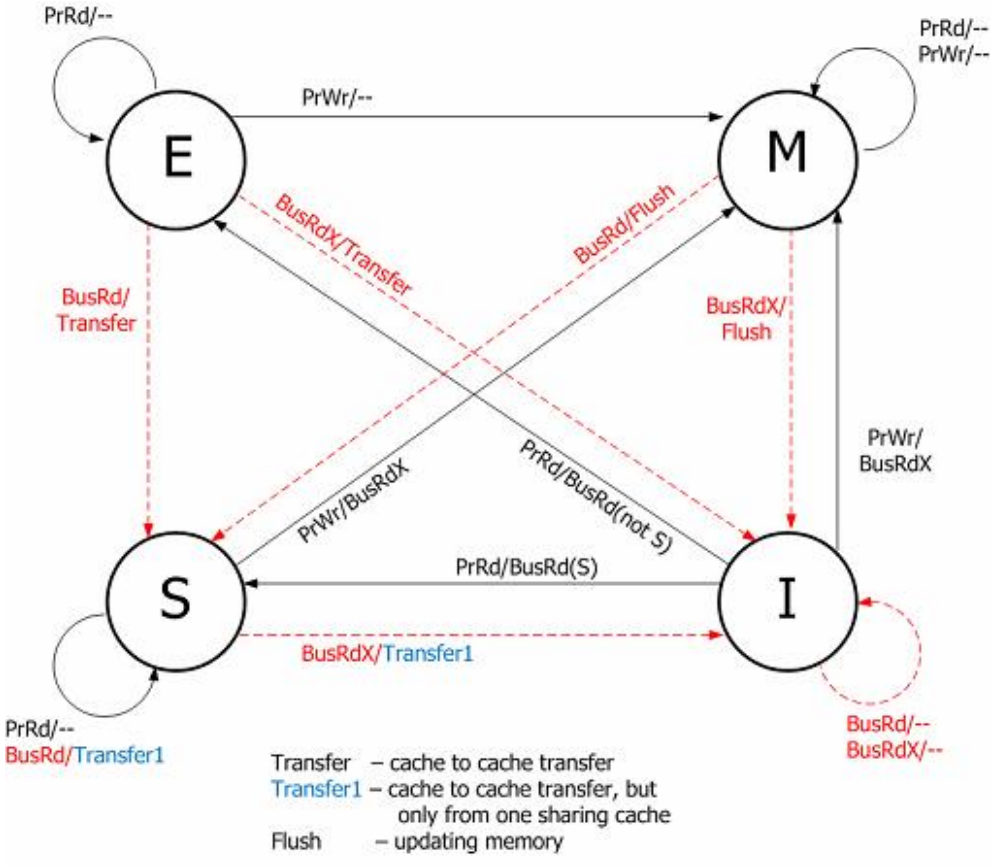
Slika 2 - MSI Dijagram Stanja

2.3. MESI

MESI je protokol koji je nastao evolucijom MSI protokola. MESI uvodi novo stanje *Exclusive*. Keš linija se nalazi u stanju *Exclusive* kada je to jedini keš sa datim podatkom i kada se sadržaj podatka u kešu poklapa sa podatkom u memoriji. Za njegovu hardversku implementaciju, potreban je dodatni indikator na magistrali, čiji sadržaj je na animaciji predstavljen dodatnom deljenom (engl. *shared*) linijom. Tu liniju aktiviraju keševi drugih procesora, kada se pri operaciji upisa ili čitanja na magistrali nađe adresa podatka koji se nalazi u nekom od tih keševa. Prilikom učitavanja podatka u keš liniju iz memorije, ukoliko nijedan drugi keš nije postavio *shared* liniju na magistrali, stanje učitane keš linije se postavlja na *Exclusive*. Kada neki drugi keš vrši čitanje podatka koji se nalazi u liniji označenoj sa *Exclusive*, stanje *shared* linije se menja na aktivnu vrednost, a podatak za drugi keš se dovlači iz memorije i takođe se postavlja u stanje *Shared*. U stanjima *Shared* i *Exclusive* memorija je ažurna.

Prilikom upisa u keš liniju koje je u *Exclusive* stanju, ne vrši se poništavanje jer za tim nema potrebe, već ta linija samo prelazi u stanje *Modified*, bez izlaska na magistralu i ažuriranja memorije. Prilikom čitanja ili upisa, ukoliko se podatak nalazi u nekom kešu u stanju *Modified*, taj keš je dužan da dostavi podatak kešu koji ga zahteva i pritom ažurira memoriju. Prilikom zamene *Modified* bloka potrebno je izvršiti ažuriranje bloka u memoriji. Postoje razlike u implementacijama ovog protokola za slučaj čitanja podatka koji se u drugom kešu nalazi u stanju *Shared* ili *Exclusive*, a odnose se na to odakle se dobija podatak koji se traži. U slučaju da se podatak nalazi u *Exclusive* stanju u nekom drugom kešu, podatak se prosleđuje iz tog keša (*cache supply*), a moguće je varijanta da se podatak dovlači iz memorije (*memory supply*). Ako se podatak nalazi u jednom ili više drugih keševa u stanju

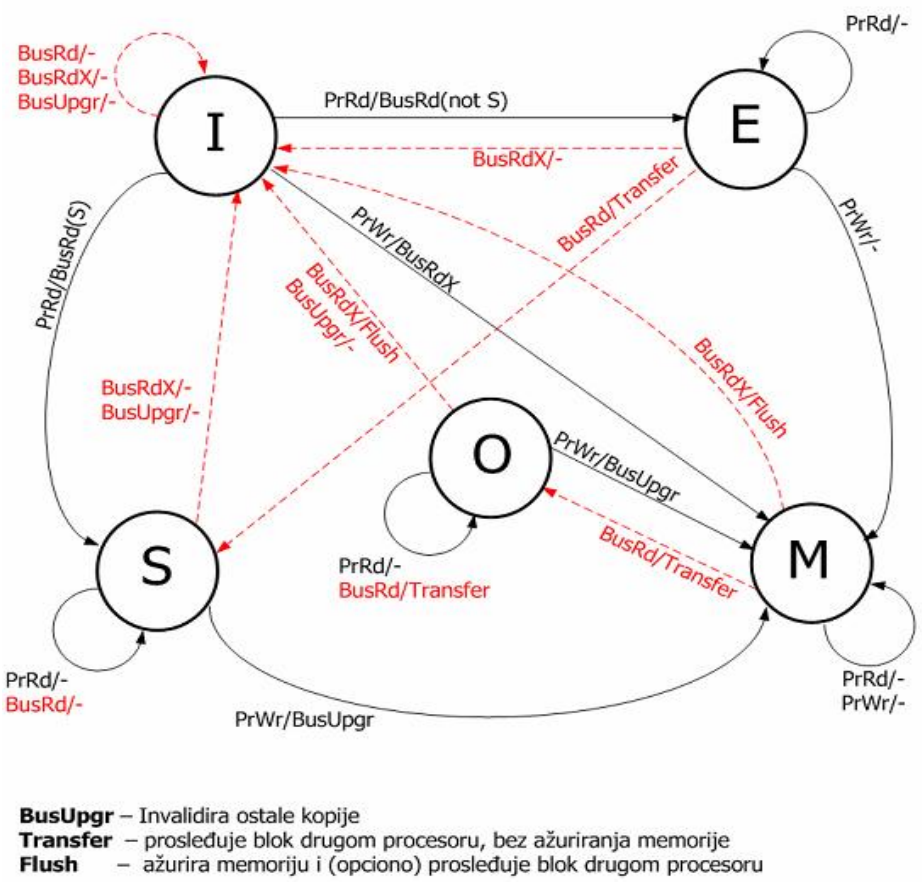
Shared, moguće su varijante da samo jedan od njih prosledi podatak drugom kešu preko magistrale podataka (u kom slučaju je potrebna arbitracija) ili, da svi koji ga imaju postave taj podatak na magistralu podataka, u kom slučaju je potrebno da magistrala bude projektovana u tehnologiji koja dozvoljava da više procesora postavi isti podatak istovremeno (svi ti podaci su identični). Treća mogućnost je da se podatak čita iz memorije, jer je i ona ažurna ako neko ima podatak u stanju *Shared* ili *Exclusive*. Prilikom upisa, svaki keš koji detektuje kopiju tog podatka kod sebe mora je poništiti (pri tom, oni koji treba da urade *cache supply* dužni su da postave taj podatak na magistralu)



Slika 3 - MESI Dijagram Stanja

2.4. MOESI

MOESI je nastao evolucijom MESI protokola. MOESI uvodi stanje *Owned*. Podaci koji se nalaze u keš liniji koja se nalazi u stanju *Owned* nisu isti kao podaci u memoriji, odnosno podaci u memoriji se smatraju neažurnim, ali se identični podaci (ažurni) takođe mogu nalaziti i u keš linijama drugih procesora, i to u stanju *Shared*. Prilikom dovlačenja podatka sa magistrale u keš liniju, procesor koji dovlači podatak taj podatak dohvata iz druge keš linije umesto iz memorije, ukoliko je ta linija u stanju *Modified, Exclusive* ili *Owned*. Tom prilikom nema ažuriranja memorije. Samo jedan keš može imati liniju u jednom od ova tri stanja u jednom trenutku. Podatak se upisuje u memoriju tek u slučaju izbacivanja podatka iz keša (engl. *flush*) koji je u stanju *Modified* ili *Owned*. Savremeni AMD64 procesori koriste ovaj protokol [4]. Ostatak protokola je identičan MESI protokolu,



Slika 4 - MOESI Invalidacioni protokol

3. Ažurirajući protokoli

Za razliku od poništavajućih, ažurirajući protokoli štede cikluse na magistrali koji su potrebni za ponovno preuzimanje čitavog keš bloka iz memorije ili iz drugog keša, tako što kada detektuju upis u memoriju na magistrali, umesto da proglase keš liniju nevažećom (engl. *invalid*), jednostavno modifikuju sadržaj keš linije tako da oslikava tačno, aktuelno stanje.

3.1. *Dragon*

Dragon protokol razlikuje keš linije prema dvema osnovnim karakteristikama. Za svaku keš liniju bitno je zapaziti da li ona sadrži ekskluzivnu kopiju nekog bloka, i da li je ta kopija u međuvremenu modifikovana ili ne. Samim tim neka keš linija može biti u jednom od sledeća 4 stanja:

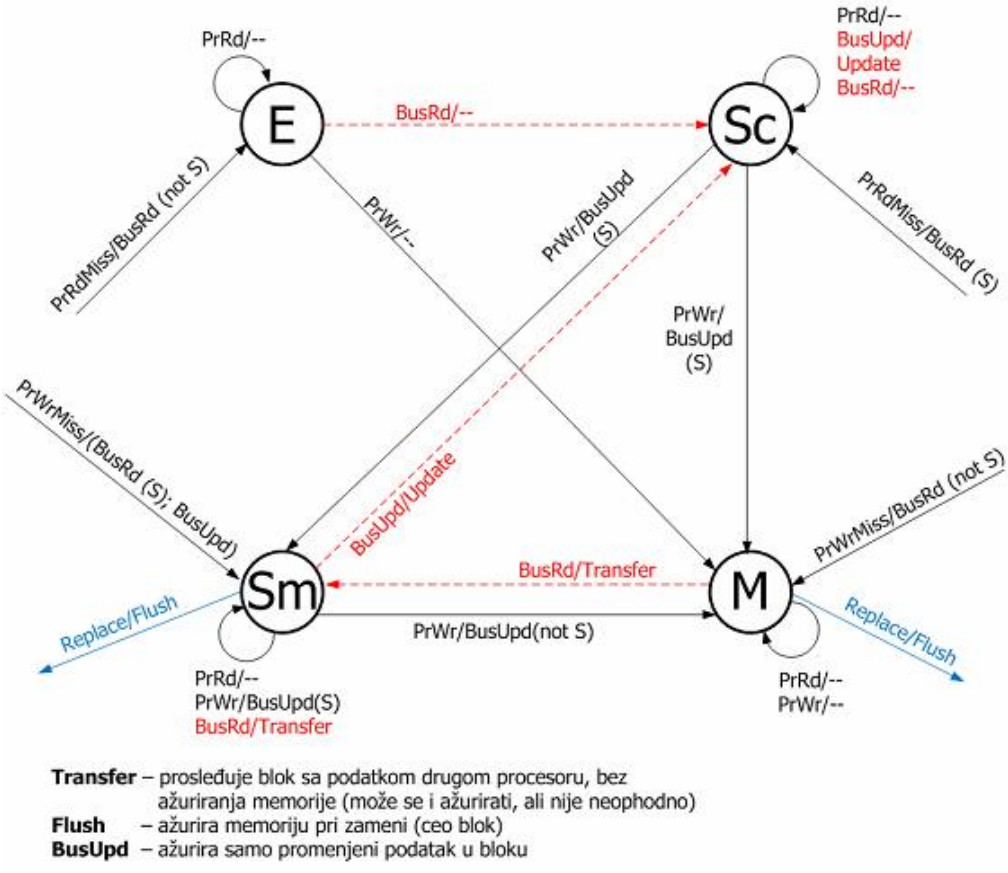
- Ekskluzivna čista kopija (engl. *exclusive clean – E*) označava stanje u kome samo ova keš linija ima validnu kopiju bloka koji nije modifikovan.
- Ekskluzivna modifikovana kopija (engl. *modified – M*) označava stanje u kome keš linija je važeća i modifikovana, a sadržaj bloka u memoriji ne odgovara vrednosti keš linije (zastareo je) i odgovornost je procesora da, prilikom zamene tog bloka drugim blokom, upiše važeću vrednost iz keš memorije u operativnu memoriju.
- Deljena modifikovana kopija (engl. *shared modified – Sm*) označava stanje u kome potencijalno 2 ili više procesora imaju blokove u svojim keš memorijama, blok u memoriji nije validan, i odgovornost je procesora da osveži memoriju kada se blok zameni u kešu. U svakom momentu, samo jedna keš linija u čitavom sistemu može biti u ovom stanju.
- Deljena čista kopija (engl. *shared clean – Sc*) označava stanje u kome keš linija potencijalno (ali ne i obavezno) može deliti blok sa drugim procesorima, pri čemu nije garantovano ni da je podatak u memoriji važeći, jer to zavisi od postojanja *Sm* keš linije sa tim blokom u nekom drugom kešu. Ako postoji *Sm* keš linija sa tim podatkom, to znači da memorija ima zastareo podatak i mora se ažurirati prilikom zamene *Sm* bloka. Ako *Sm* keš linija ne postoji, onda je memorija ažurna.

Treba napomenuti da zbog toga što je *Dragon* ažurirajući protokol ne postoji nevažeće (engl. *invalid*) stanje, jer protokol garantuje ispravnost podataka. Međutim, ako blok uopšte nije prisutan u kešu, može se smatrati da je u specijalnom nevažećem ili nepostojećem stanju.

Zahtevi procesora, transakcije na magistrali i sve akcije *Dragon* protokola su slične MESI protokolu. Za procesor se smatra da i dalje izdaje zahteve za čitanje (*Rd*) i upis (*Wr*).

Od transakcija na magistrali postoje, čitanje (*BusRd*), ažuriranje (*BusUpd*). Ažuriranje (*BusUpd*) je nova transakcija koja uzima određenu reč od strane procesora i objavljuje je (*broadcast*) na magistrali tako da se svi keševi drugih procesora ažuriraju (ali ne i memorija). Time što se objavljuje samo sadržaj specifične reči umesto čitavog bloka, teži se efikasnijem iskorišćenju magistrale. Kao i kod MESI protokola, postoji deljena linija (*shared*) na magistrali koja omogućava keš kontrolerima da znaju kada datu keš liniju dele sa drugim keš memorijama. Operacija *Transfer* prosleđuje podatak iz jednog keša drugom kešu prilikom zahteva za čitanje, i pritom nema potrebe da ažurira memoriju (mada neke implementacije ažuriraju, zbog lakše realizacije logike protokola).

Konačno, što se tiče akcija, jedina nova mogućnost je da keš kontroler ažurira lokalnu keš liniju (*Update*) podacima koje dobija preko magistrale od drugog keša koji u tom trenutku radi *BusUpd* transakciju.



Slika 5 - Dragon Dijagram stanja

Na prethodnoj slici, prikazana je osnovna struktura *Dragon* protokola. Iz ugla procesora mogu se posmatrati sledeća stanja i adekvatne akcije koje se dešavaju kada se desi promašaj čitanja, pisanja ili uspešno pisanje.

Promašaj čitanja. Generiše se transakcija čitanja magistrale (*BusRd*). U zavisnosti od statusa deljene linije, blok se učitava u keš liniju kojoj se dodeljuje *E* ili *Sc* stanje. Konkretnije, ako je blok u *M* ili *Sm* stanju u nekom od drugih keševa, taj keš proverava stanje deljene linije i dostavlja validne podatke za blok na magistrali. Dalje se taj blok učitava u lokalni keš u okviru *Sc* stanja. Ako je drugi keš bio u stanju *M*, njegovo stanje se sada menja u stanje *Sm*. Ako ni jedan drugi keš nema kopiju, i

samim tim deljena linija nije aktivna, podaci se dostavljaju iz memorije i blok se učitava u lokalni keš u stanje *E*.

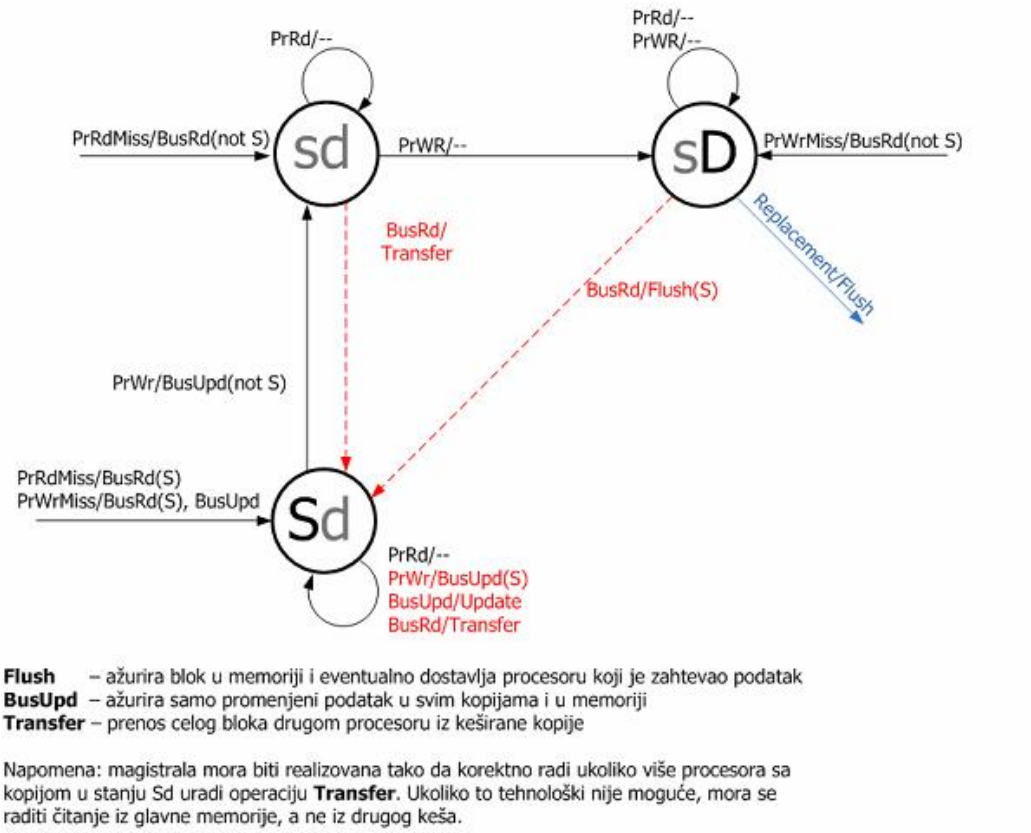
Upis. Ako je blok bio u *Sm* ili *M* stanju na lokalnom kešu, onda se ne vrše nikakve akcije. Ako je blok bio u *E* stanju, onda ta linija prelazi u *M* stanje. Ako je blok bio u *Sc* stanju, generiše se transakcija ažuriranja magistrale (*BusUpd*). Ako bilo koji drugi keš ima kopiju podataka, onda on ažurira svoju keširanu kopiju, i postavlja svoje stanje na *Sc*. Lokalni keš takođe ažurira svoju kopiju bloka i menja svoje stanje u *Sm*. Ako ni jedan drugi keš nema kopiju podatka, i samim tim deljena linija nije aktivna, lokalna kopija se ažurira i stanje prelazi u *M*. Konačno, ako prilikom upisa, blok se ne nalazi u kešu, operacija upisa se ponaša kao promašaj čitanja nakon koga ide uspešan upis tj. prvo se generiše transakcija čitanja magistrale (*BusRd*) a onda ako je blok nađen u drugim keš memorijama transakcija ažuriranja magistrale se generiše (*BusUpd*).

Zamena. U slučaju kada je neophodno zameniti blok u okviru jedne keš linije drugim (nije prikazano na dijagramu), blok se upisuje nazad u memoriju koristeći transakcije magistrale samo ako je u *M* ili *Sm* stanju. Ako je u *Sc* stanju onda ili neki drugi keš (u stanju *Sm*) je odgovoran za upis ili takvih nema što znači da je vrednost u memoriji već ažurna.

3.2. *Firefly*

Jedna modifikacija *Dragon* protokola koja ima za cilj da poboljša performanse za konkretan slučaj korišćenja je *Firefly*. Uvidelo se da je moguće eliminisati "Deljeno modifikovano stanje" (*Sm*) time što bi se prilikom svake transakcije ažuriranja magistrale, mogla ažurirati i glavna memorija. Razlog zašto ovaj koncept nije iskorišćen kod *Dragon* protokola je što se podrazumevalo da keševi i glavna memorija koriste različite memorije (SRAM i DRAM respektivno), što je značilo da su se keš memorije mnogo brže ažurirale od glavne memorije i samim tim bi bilo kontraproduktivno da se memorija ažurira pri svakoj transakciji ažuriranja. Pored ove razlike, uvedena je još jedna promena, postavlja se pitanje da li je neophodno obaveštavati druge keš memorije prilikom izbacivanja jednog *Sc* bloka, tako da ako ostane samo jedan *Sc* blok da on pređe u *E* ili *M* stanje? Iako sve implementacije *Firefly* protokola nemaju ovo implementirano, ova karakteristika poboljšava performanse, jer predstavlja sistem koji komfornije odgovara realnom stanju blokova i samim tim smanjuje šanse da dođe do kolizije i forsirane serijalizacije između zahteva za ažuriranje na magistrali i eventualnog čitanja/upisa u memoriju.

Ekvivalenti stanjima *E*, *Sc*, *Sm* i *M* u *Firefly* protokolu *su sd, Sd, sD* (*shared* & *dirty*, sa velikim slovom kod onog stanja koje ima tu osobinu). Simulirana modifikacija Firefly protokola se od Dragon protokola razlikuje po tome što *BusUpd* operacija ažurira podatak i u memoriji, a ne samo u deljenim kopijama, pa stanje *Sm*, odnosno *SD* postaje suvišno. Takođe, prilikom upisa u *Sd* podatak (*Shared clean*), ne ide se u stanje *sD* (*Modified*), već se ostaje u istom stanju ako je taj podatak i dalje deljen (linija S na magistrali je i dalje aktivna u trenutku upisa). Ako podatak više nije deljen (linija S na magistrali nije aktivna u trenutku upisa), tada se ide u stanje *sd* (*Exclusive*). To je moglo biti urađeno jer *BusUpd* operacija u trenutku upisa ažurira i memoriju u ovom protokolu. Jedino stanje kada memorija neće biti ažurna je kada u stanju *Exclusive* dođe do upisa od strane tog procesora. Tada će podatak preći u stanje u stanje *Modified*, bez akcija na magistrali. Izmenjena vrednost će sigurno biti ažurirana u memoriji – ili operacijom *flush* u slučaju zamene datog podatka ili operacijom *BusUpd* u slučaju pristupa tom podatku od strane nekog od ostalih procesora.



Slika 6 - Dijagram stanja *Firefly* protokola

4. Vivio simulacije

Simulacije su zasnovane na postojećim radovima MESI i *Firefly* simulacija. Uvedene su određene modifikacije nad postojećim simulatorima i napravljeni su nedostajući za ostale protokole pomenute u okviru ovog dokumenta.

Svaki od simulatora je vizuelno predstavljen sa tačno određenim brojem procesora (podrazumevano 4) od kojih svaki može da čita ili piše u jednu od 4 memorijske lokacije dostupne u okviru glavne memorije. Svakom procesoru odgovara tačno jedan keš kontroler koji u sebi sadrži dve linije za podatke u keš memoriji. Ukoliko postoje samo dve keš linije, jedna od njih odgovara parnim, a druga neparnim memorijskim adresama. Predstavljene su adresna magistrala i magistrala podataka, kao i deljena linija kontrolne magistrale. Pritiskom na odgovarajuće dugmiće na svakom od procesora mogu se izdavati komande paralelno, čime se daje utisak ponašanja realnog sistema prema protokolu te simulacije.

Sam programski jezik podržava koncept klasa, i rad sa više niti upotrebom *fork* funkcije.

4.1. Model, ključne klase i metode

Izvorni kod simulacije u vidu skripte za Vivio simulator je organizovan, što je više bilo moguće u skladu sa objektno orijentisanom paradigmom. U okviru izvornog koda, sve elemente možemo podeliti grubo u dve grupe. One koji su zaduženi za iscrtavanje grafičkih elemenata na ekranu i na logičke elemente koji implementiraju logiku protokola.

Upotrebljene klase su ukratko opisane u sledećem tekstu.

`BusArrow`. Prvenstveno vizuelna klasa koja služi da vertikalno animira tok podataka, obično se koristi u okviru neke od logičkih klasa. Koristi se između procesora i keša, keša i magistrale, magistrale i memorije. Od značaja su metode ***`reset`***, ***`moveup`*** i ***`movedown`***.

`Memory`. Sadrži neophodne elemente za vizuelni prikaz sadržaja memorije, kao i sam sadržaj memorije na odgovarajućim adresama. Od istaknutijih metoda, tu su ***`highlight`*** i ***`reset`*** koje se koriste za senčenje trenutno aktivnih memorijskih lokacija.

`Bus`. Isključiva namena je prikaz horizontalnih bus linija. Sadrži metodu ***`highlight`***.

`Cache`. Logika i implementacija *cache-coherence* protokola. Ima nekoliko metoda od značaja.

- **`flush`** – Upisuje sadržaj keš linije u memoriju i prazni keš liniju.
- ***`watch`*** ili ***`shared`*** – Kao argumente prima sadržaj magistrale i reaguje u zavisnosti od implementiranog protokola. Prilikom izlaska podešava globalne promenljive koji imaju ulogu semafora.
- ***`buswatch`*** ili ***`sharedwatch`*** – Koristi se poput statičke metode u drugim programskim jezicima. Ima za cilj da sinhrono pozove ***`watch`*** metodu svih instanci keševa, izuzev onog koji je započeo transakciju, i, u zavisnosti od reakcije ostalih keševa, radi adekvatne operacije nad memorijom.
- ***`read`*** – Keš je dobio od procesora zahtev za čitanjem, i procesira ga u skladu sa protokolom.
- ***`write`*** – Keš je dobio od procesora zahtev za upisom, i procesira ga u skladu sa protokolom.

`CPU`. Ova klasa ima za cilj prvenstveno prikaz dugmića kojima korisnik može da izdaje operacije čitanja i upisa.

5. Pravci daljeg razvoja

Pored implementacije i adaptacije novih i neobrađenih protokola (*MESIF* [5], [6], *Write-Once*), od značaja za pomoć pri nastavi je mogućnost pravilne serijalizacije i beleženja operacija. Trenutno, konkurentni zahtevi za magistralom se rešavaju uz pomoć brave (engl. *mutex*) po procesoru koji pristupa magistrali. Bolje rešenje bi bilo, kada bi se svi zahtevi od korisnika ređali u FIFO red (engl. *queue*), jer bi tako korisnik bio u mogućnosti da zadaje i po nekoliko komandi zaredom sa istog procesora, koje bi imale garantovan redosled izvršavanja, što je korisno ukoliko postoje predefinisane sekvence primera koje korisnik hoće da proba.

Takođe, bilo bi korisno da se na svim simulacijama doda po jedan vizuelni element koji bi predstavljao spisak svih izvršenih instrukcija u formatu sličnom onom koji je upotrebljen u ostalim nastavnim materijalima (npr. **`1. P0,R,A0`**).

6. Zaključak

Vivio se zbog lakoće programiranja pokazao kao veoma koristan alat za vizuelizaciju rada mnogih algoritama i protokola. Zbog svoje podrške za klase i niti moguće je u njemu napraviti kompleksne vizuelne simulacije na lak način, i korišćenjem prethodno razvijenih elemenata (engl. *code reuse*). Jedina mana je što je baziran na *ActiveX* tehnologiji i kao takav zavisi od *Windows* operativnog sistema. Takođe, utvrđeno je da zavisi i od arhitekture sistema na kom se izvršava i trenutno je ograničen na 32bitne arhitekture.

7. Reference

[1] Internet stranica Vivio okruženja,
`https://www.cs.tcd.ie/Jeremy.Jones/vivio/vivio.htm`

[1.1] Vivio simulacija MESI protokola,
`https://www.cs.tcd.ie/Jeremy.Jones/vivio/caches/MESI.htm`

[1.2] Vivio simulacija *Firefly* protokola,
`https://www.cs.tcd.ie/Jeremy.Jones/vivio/caches/firefly.htm`

[2] *Parallel Computer Architecture, A Hardware / Software Approach, David Culler, Jaswinder Pal Singh, Anoop Gupta,* 1997.

[3] Opis ponašanja protokola koherencije keš memorije,
`http://en.wikipedia.org/wiki/Cache_coherence`

[4] *AMD64 Architecture Programmer's Manual Vol. 2: Systems Programming,*
`http://www.amd.com`
`/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf`

[5] *Coherency Leaps Forward at Intel*,
http://www.realworldtech.com/page.cfm?ArticleID=RWT082807020032&p=5

[6] *Forward state for use in cache coherency in a multiprocessor system*,
http://www.freepatentsonline.com/6922756.html

8. Spisak izmena na dokumentu

- Andrija Bošnjaković, ispravljene uočene greške, novi dijagrami, 01.02.2009. 20:07
- Andrija Bošnjaković, revizija pred objavljivanje, 30.11.2008. 14:12
- Đorđe Jevđić, ispravljene uočene greške, novi dijagrami, 28.11.2008. 18:29
- Andrija Bošnjaković, revizija pred objavljivanje, 25.2.2008. 18:12
- Aleksandar Stevanović, konačna verzija za predaju uz projekat, 24.2.2008. 22:53
- Srđan Rosić, organizacija dokumenta, dodatni materijal, 20.2.2008. 18:32
- Srđan Rosić, dorađeni dijagrami, 18.2.2008. 00:40
- Aleksandar Stevanović, prva verzija, 17.2.2008. 22:25