**Memorie Transacțională Conștientă de Dependențe pentru Creșterea Concurenței**

**-referat-**

Hany E. Ramadan, Christopher J. Rossbach, and Emmett Witchel Department of Computer Sciences University of Texas at Austin Austin, TX, USA

Ideea principală a DATM (Dependence-Aware Transactional Memory) este de a permite orice intercalare a execuțiilor tranzacțiilor care este serializabilă în context de conflicte, inclusiv intercalări ce implică conflicte simple. Spre deosebire de sistemele de memorie tranzacțională (TM) convenționale, care reduc concurența utilă reluând tranzacțiile conflictuale chiar și atunci când intercalările lor ar fi serializabile, DATM gestionează activ dependențele dintre tranzacțiile nefinalizate. În anumite cazuri, aceasta presupune transferul de date între tranzacții, facilitând comiterea lor sigură. Evaluarea prototipului a demonstrat că DATM crește concurența și eficiența, reducând timpul de execuție al benchmark-urilor STAMP cu până la 39% și numărul de reluări ale tranzacțiilor cu până la 94%.

Memoria tranzacțională reprezintă o abstracție inovatoare pentru simplificarea programării concurente, ajutând programatorii să utilizeze mai eficient arhitecturile moderne de calcul paralel. Aceasta asigură execuția atomică și izolată a secțiunilor critice, în care atomicitatea garantează că o tranzacție fie finalizează complet, fie este anulată integral, iar izolarea previne vizibilitatea stărilor parțiale ale tranzacțiilor în curs. Astfel, fiecare tranzacție produce efecte liniare, care par a fi aplicate instantaneu între începutul și sfârșitul său.

Conflictele tranzacționale apar atunci când două sau mai multe tranzacții accesează aceleași date, cel puțin una dintre ele încercând să le modifice. Execuțiile sunt serializabile în context de conflicte dacă ordinea accesărilor conflictuale respectă o ordine logică serială a tranzacțiilor. Implementările TM actuale, însă, se bazează pe reluarea sau blocarea tranzacțiilor în caz de conflict, ceea ce limitează concurența. Acestea tratează implicit datele accesate de o tranzacție ca fiind „blocate”, generând astfel conflicte suplimentare.

DATM introduce o abordare inovatoare printr-o memorie tranzacțională conștientă de dependențe, capabilă să asigure serializabilitatea în context de conflicte fără a penaliza concurența. Tranzacțiile devin conștiente de dependențe, iar valorile pot fi transferate între tranzacțiile nefinalizate atunci când este necesar. Această strategie permite comiterea în siguranță a tranzacțiilor conflictuale care sunt serializabile, crescând eficiența și utilizarea hardware-ului paralel. În plus, abordarea DATM este complet transparentă pentru programatori, aspect esențial având în vedere problemele de performanță ale structurilor de date partajate, precum contoarele sau listele înlănțuite, în sistemele TM convenționale.

**Contribuțiile lucrării:** Lucrarea introduce tranzacțiile conștiente de dependențe, un model TM care permite un nivel semnificativ mai mare de concurență comparativ cu metoda tradițională de blocare în două faze. Totodată, propune un design hardware pentru un sistem TM conștient de dependențe, bazat pe un protocol de coerență a cache-ului denumit FRMSI (Forward/Receive MSI). Implementarea prototipului a fost evaluată folosind seturi de benchmark-uri precum STAMP și TxLinux, demonstrând îmbunătățiri notabile în performanță.

**Creșterea concurenței cu DATM** Modelul conștient de dependențe creează și urmărește dependențele dintre tranzacțiile care accesează aceleași date, permițând în unele cazuri transferul speculativ al datelor între tranzacții. Aceste dependențe permit ca DATM să comită tranzacții care ar fi fost reluate sau blocate de un TM convențional, utilizând mai bine munca concurentă.

**Exemplu: Contor partajat**. Autorul foloseste notația standard pentru dependențele de date, cum ar fi W→R, care indică faptul că o celulă de memorie a fost scrisă de o tranzacție și apoi citită de alta. În acest context, o „celulă de memorie” este o linie de cache, dacă nu se specifică altfel. DATM stabilește o dependență W0→R1 pentru contor și se asigură că tranzacția T1 este comisă după T0. Astfel, tranzacțiile pot continua concurent, chiar dacă ambele scriu în aceeași locație de memorie.

Execuția este serializabilă în context de conflicte, dar nu ar fi permisă de stilul de detecție al conflictelor bazat pe blocarea în două faze, folosit în sistemele TM actuale. În acest caz, accesul la contor de către T1 după ce a fost scris de T0 este considerat un conflict, ceea ce ar forța blocarea T1 sau reluarea unei tranzacții. O execuție însă neserializabilă esre una in care ambele tranzacții nu pot fi comise cu succes. DATM detectează acest ciclu de dependențe și anulează una dintre tranzacții pentru a-l elimina.

**Acceptarea mai multor intercalări** nu garantează automat performanțe mai bune față de abordările convenționale, deoarece mulți alți factori influențează performanța. Totuși, DATM crește probabilitatea utilizării resurselor paralele atunci când tranzacțiile rulează concurent: în loc să intre în conflict, tranzacțiile pot coopera și pot fi comise ambele.

**Compararea cu alte strategii de rezolvare a conflictelor**. DATM și alte sisteme execută un set de tranzacții care intră în conflict pe un singur element partajat. DATM creează o dependență de la T1 la T2, fără a forța blocarea sau reluarea vreuneia dintre tranzacții. Astfel, T2 este comisă mai devreme decât în alte strategii de rezolvare a conflictelor, datorită acceptării intercalărilor de acces care ar necesita blocarea sau reluarea în alte sisteme.

**Modelul creat:** Dacă există o dependență XA→XB, tranzacția A trebuie să fie comisă înainte de B. Sistemul urmărește toate dependențele la nivel de linii de cache, creând noi dependențe între tranzacții ca răspuns la accesările memoriei la timpul de execuție. Ordinea tranzacțiilor depinde de comportamentul lor dinamic. În cazul unei dependențe W→RW, sistemul transferă datele din linia de cache la momentul creării dependenței, marcând linia ca fiind transferată. Pentru o dependență W0→R1, W0 -> R1, tranzacția T0 este sursa, iar T1 este destinația sau tranzacția dependentă.

Pentru a menține serializabilitatea, o tranzacție dependentă poate citi o valoare de la o tranzacție sursă doar dacă aceasta reprezintă valoarea finală a liniei de cache pentru tranzacția sursă. Dacă tranzacția sursă suprascrie datele transferate, tranzacția destinație trebuie să fie reluată. Dependențele sunt create pentru fiecare linie de cache la prima accesare, iar accesările ulterioare la același obiect nu afectează structura dependențelor. De exemplu, dacă T0 scrie într-o linie de cache, apoi T1 scrie și citește linia respectivă, dependența rezultată este W0→W1. Când o tranzacție este comisă sau anulată, toate dependențele acesteia dispar.

**Dependențe multiple** Ele apar atunci când două tranzacții intră în conflict pe mai multe linii de cache. Fiecare linie de cache generează o dependență separată. În cazul dependențelor multiple, relația dintre tranzacții este guvernată de cea mai restrictivă dependență din fiecare direcție. De exemplu, W→R este mai restrictiv decât W→ W sau R→W, iar acestea din urmă nu sunt ordonate relativ una față de cealaltă. Dacă mai mult de două tranzacții accesează simultan aceeași linie de cache, primele două creează o dependență conform regulii de mai sus, iar a treia tranzacție creează o dependență cu ultimul scriitor al liniei de cache.

**Dependențe ciclice** Toate dependențele restricționează ordinea de comitere: o tranzacție trebuie să aștepte ca tranzacția de care depinde să fie comisă. Dacă apar cicluri în graful de dependențe, lanțul ciclic poate cauza o stare de blocaj. Sistemul detectează ciclurile și relansează cel puțin o tranzacție pentru a le elimina. Reluarea tranzacțiilor încearcă să păstreze cât mai mult din munca concurentă, prioritizând tranzacțiile fără dependențe.

**Dezactivarea urmăririi dependențelor: Mod „no-dep”.** Tranzacțiile conștiente de dependențe pot coexista cu alte strategii de rezolvare a conflictelor. Dacă o tranzacție este reluată în modul „no-dep”, aceasta nu mai participă la crearea dependențelor.

**Excepții și date inconsistente:** Deoarece modelul permite transferul de date între tranzacții, o tranzacție poate citi date invalide, ceea ce poate genera excepții sau bucle infinite. Starea inconsistentă este rezolvată odată ce tranzacția sursă finalizează, iar actualizările sale provoacă reluarea tranzacției destinație. Acest proces elimină buclele infinite care nu sunt conforme cu comportamentul serial al aplicației.

**Anulări în cascadă.** Anulările în cascadă apar când anularea unei tranzacții sursă determină anularea tranzacțiilor dependente. În DATM, acest fenomen apare doar pentru dependențele W→R, în care tranzacția sursă este anulată sau suprascrie datele transferate.

**Design hardware**. DATM poate fi implementat cu un protocol nou de coerență a cache-ului numit **FRMSI** (Forward Receive MSI), alături de un vector ordonat de ID-uri ale tranzacțiilor sau o tabelă de timestamp-uri. Acest design susține ordonarea globală a comiterii tranzacțiilor și scrierilor, prevenind blocajele și dependențele ciclice. DATM adaugă două biți la cuvântul de stare al tranzacției: **ND (no-dependence)** indică faptul că tranzacția nu a intrat în dependențe. **Frc-ND (force-no-dependence)** împiedică tranzacția să creeze dependențe, forțând un mod tradițional de gestionare a conflictelor.

**Protocolul de coerență FRMSI.** Gestionarea versiunilor este complicată de transferul datelor, care permite mai multor cache-uri să modifice aceeași linie de cache. FRMSI include stări pentru transfer și primire:TMRTMR și TRTR: stări pentru liniile de cache care primesc date transferate.TMFTMF și TMRFTMRF: stări pentru liniile care au transferat date. Starea CTMCTM permite scrierea înapoi după comitere, păstrând ordinea față de alte tranzacții. Cache-urile comunică modificările prin mesaje TXOVWTXOVW care invalidează valorile speculative transferate anterior.

**Suspendarea tranzacțiilor.** DATM permite suspendarea tranzacțiilor, iar liniile de cache stochează ID-ul tranzacției pentru a permite reluarea. Orice încercare de a crea o dependență cu o tranzacție suspendată eșuează, tratând operațiunea ca un conflict.

**Cerințe de ordonare pentru DATM:**

1. Tranzacțiile dependente trebuie să comită în ordine.
2. Tranzacțiile care formează dependențe primind date speculative trebuie să devină dependente de cel mai recent scriitor al acelor date.
3. Dependențele ciclice trebuie detectate în avans și evitate prin repornirea uneia sau mai multor tranzacții.
4. Dependențele sunt tranzitive: dacă tranzacțiile sunt anulate, ordinea dependențelor trebuie păstrată pentru cele rămase active.
5. Memoriile cache care conțin aceeași linie în starea **CTM** trebuie să scrie liniile înapoi în ordinea dictată de ordinea de comitere, iar solicitările ulterioare pentru linie trebuie deservite de ultima memorie cache care a comis.

**Vectorul de ordine:** Implementarea DATM poate susține ordonarea prin menținerea unui vector de ordine al ID-urilor tranzacțiilor în fiecare cache. Fiecare cache implicat într-o tranzacție dependentă păstrează o copie a vectorului de ordine, iar toate copiile au aceleași date. Vectorul listează tranzacțiile active, sortând topologic graful dependențelor. Noi dependențe se adaugă la finalul listei. Fiecare cache trebuie să vadă dependențele în aceeași ordine. Protocolul **FRMSI** necesită mesaje suplimentare pentru extinderea la un protocol de director. **Ordinea bazată pe timestamp:** Dependențele ordonate prin timestamp folosesc un tabel de timestamp. Fiecare cache implicat într-o tranzacție dependentă menține o copie identică a acestui tabel. Fiecare intrare conține un ID de tranzacție, un timestamp, un bit de validare și un bit de activare. Ordinea timestamp-urilor se folosește pentru a preveni conflictele ciclice și a respecta dependențele. **Respectarea cerințelor de ordonare:** O tranzacție poate comite doar dacă este prima tranzacție activă în vectorul de ordine.La primirea unei linii de cache, ID-ul tranzacției se adaugă la vectorul de ordine. Hardware-ul verifică dacă tranzacția creează un ciclu și repornește o tranzacție pentru a-l rezolva.Anulările tranzacțiilor sunt semnalizate prin mesajul **xABT** pe bus, dezactivând intrările aferente din vector.**Scrierile înapoi:** Poziția în vectorul de ordine dictează atât comiterea tranzacțiilor active, cât și scrierile înapoi ale liniilor modificate. Intrările din vector rămân valide până când toate liniile asociate ies din starea **CTM**. O memorie cache notifică sistemul când finalizează ultima scriere, invalidând ID-ul din vector. **Capacitatea vectorului de ordine sau tabelului de timestamp:** Ultima intrare a vectorului trebuie să inițieze scrieri înapoi pentru liniile **CTM**.Capacitatea mare a vectorului minimizează șansele de suprasolicitare. Accesările vectorului sunt rare și sincronizate cu momentele de pierdere a datelor.

**Optimizări de performanță**: În sistemele DATM, accesările vectorului sunt rare și au loc doar în momentele critice asociate pierderii de date, fiind sincronizate pentru a asigura consistența. Cache-ul include biți de acces pentru fiecare cuvânt din linie, ceea ce permite detectarea rapidă a actualizărilor pe bus. În funcție de aceste modificări, se decide fie reluarea tranzacțiilor, fie actualizarea liniei cache. În plus, fiecare procesor menține un set de tranzacții predecesoare (PredSet), care reduce blocajele generate de tranzacții lungi asupra celor scurte. Tranzacțiile pot fi comise chiar dacă nu sunt primele în ordinea vectorului, cu condiția ca setul lor de predecesori să fie gol.

Evaluarea performanței DATM s-a realizat utilizând un simulator HTM într-o configurație SMP cu 16 procesoare. Rezultatele arată că, în scenarii cu concurență ridicată, cum ar fi **bayes** și **vacation**, DATM reduce semnificativ numărul de restarturi și îmbunătățește performanța cu până la 39%. În sarcinile cu conflicte reduse, impactul asupra performanței este minim. Dependențele gestionate eficient permit o mai bună concurență, reducând numărul de reluări și optimizând execuția generală.

**Constrângeri hardware:** Granularitatea pe linie de cache influențează traficul pe bus, însă granularitatea pe cuvinte oferă o precizie superioară, reducând conflictele. Gestionarea conflictelor prin prioritizarea tranzacțiilor cu mai puține dependențe minimizează atât blocajele, cât și aborturile în cascadă, ceea ce contribuie la stabilitatea sistemului.

**În concluzie** DATM demonstrează că utilizarea tranzacțiilor conștiente de dependențe poate îmbunătăți semnificativ performanța sistemelor HTM printr-o execuție mai concurentă. Modelul se dovedește eficient și complet transparent pentru utilizatori, eliminând necesitatea modificărilor în programe. Această abordare reprezintă un progres important față de modelele HTM tradiționale, oferind o soluție performantă pentru aplicațiile paralele.