• mai avem în vedere faptul că sist. tb. creat atît µP din stg. să necesite acces . . . .

SPM.20 pg.25: comutator crossbar: ∃ atîtea bare oriz. cîte µPs;
— — vertic. cîte memorii;
la ∩ barelor oriz. & vertic. = comutat. unidir: µP → mem.;
comut. tb. să fie bidir. ptr. conect. liniilor de dte ale µP.:
RW, WR – inactive ⇒ comut. sunt în starea 3state;
RW activ, WR inactiv ⇒ sens de curgere a dtelor: e mem. → µP.;
RW inactiv, WR activ ⇒ — : µP. → mem.;
în (t) mom. de timp (t) µP. poate fi conect. la un modul de mem. (cînd ∃ tot atîtea module de mem. cîte µPs. și nici o pereche de µPs. nu solicită acces la mem. simultan; altfel, doar unuia i se dă acces la mem., celălalt așteaptă);
crește lățimea de bandă (teoretic ar crește de n ori dc. sist. are n proc. față de sist. cu mag. unică);

dezavantaje: dificultatea realiz. comutatoarelor
liniile de adr. ⇒ buffere 3state unidir.
liniile de dte. ⇒ 2 buffere 3state unidir. din care doar unul e în st. activă, celălalt e în 3state;
"●" → circuite complexe cu dezav. ⇒ cost, fiabilitate redusă, nr. mare de pini, nr. mare de lipituri (cauza fiț. defectoase)

→ tehnologia GBA (Grid Ball Array)

pg.27: latența de comutare = timp scurs între cererea de comut. și comut. propusă

pg.28: "met. grăună moartă" → utiliz. memorii multiport ⇒ comutatoarele sunt mai simple însă e complicată realizarea memoriei multiport
→ nu pot acorda acces simultan ptr. 2 procesoare la ac. modul de mem.;
→ avantaj: ac. mem. se constr. prin integr. directă însă cîte µPs. sunt, atîtea porturi de mem. tb. să am;
→ dezavantaj: nr. f. mare de pini;

pg. 34: ex. istoric de sist. multiprocesor (istoric)

pg. 35: KSR1 → utiliz 1088 μPs pe 64 biți;
→ sp. virt. de mem. com. 2^{40} octeți;

- toate dtle. sist. sunt distribuite prin inelele de niv. 0 în mem. cache
- μP. grupate în clustere ptr. că μPs. folos. în comun setul de dte. pe inelul căruia îi sunt asociate;
- de. un μP. necesită seturi de dte. care nu sunt pe inelul său (cache local), prin Group Directory se det. pe care inel se află set → acces;

pg. 39: Asigurarea coerenței mem. cache în sist. cu μPs. multiple:
→ ∃ o mem. comună în care se află elem. x; el stă aici până când Pa cere acces la mem. comună ptr. a copia x de aici în local cache; de. Pa, Pb citesc elem. x în ||, e în regulă; de însă Pa face o modif. în $x \to x'$ ⇒ coerența se pierde ptr. că Pa lucrează cu forma actualizată a dtlor x' iar Pb lucrează cu dtle. vechi;
→ ar fi bine, ca atunci când a apărut modif., ea să fie făcută cunoscută celor care utiliz. elem. x;
→ tehnica Read Through: un μP lucrează cu refill line ce conține x atâta timp cât are nevoie; în acest interval → incoerență; la un mom. dat refill line fi golit ptr. a fi umplut cu altceva la golire, info. e actualizată în mem. principală;
→ Write Through: ori de câte ori se face modificarea lui x, se face imediat actualizarea în mem. principală;
→ tehnica Write Through este mare consumatoare de timp; la multi μP. Pa nu știe că Pb dorește să utilizeze elem. x care a suferit modificări ⇒ când Pa modifică x, modificarea fi anunțată la întreg sist. ptr. ca info. să nu fie eronată;

2 modalități: • write update: - pp. că op. de scrie care modif. pot fi urmată de actualizarea lui x în toate locațiile în care ac. info. ∃;
- fi făcută modif. și la mem. comună;

•• write invalidate: – este mai radicală;
– dc. Pa modifică elem. de dte.

$X \rightarrow X'$ ceea ce mai are de făcut ptr. coerență e să invalideze toate celelalte copii ale lui X din sist. ai proc. care vor să modifice X să vadă că nu se poate;

→ avantaj: în loc să se copieze X de n ori, - - -

- - -

→ dezavantaj: n=72 μPs., dc. toate au nevoie de X' ⇒ tb. să îl ia din mem. cache a μP. care a făcut modif. ⇒ consum de timp;

pg. 44: Scheme { hw. → rapide, rigide; sw. → flexibile, lente; combinate → OK }

Protocoale ptr. Write Update / Write Invalidate:

→ fac parte din categ. snoopy cache protocols

1. Se difuzează pe mag. info. ce tb. actualizată. μP. "adulmecă" info. difuzată, iar cele care lucrează cu info. respectivă actualiz. info. respectivă.

2. Alte sist. nu pot face difuzarea. Atunci ptr. fiecare elem. de dte. avem un set de pointeri care arată mem. cache ale căror μPs. utilizează info. ⇒ se analizează lista de pointeri → se det. ce mem. cache tb. să actualizeze info.;
Nu se mai distribuie tuturor μPs., ci doar celor care fol. info.
Pb. legate de pointeri → câți pointeri?
→ tb. introdus un bit de stare "dirty bit" calificat care spune dc. dtle nu sunt actualizate (setat 1); dc. dirty bit=0, dtle. conectate tb. transmise mem. ptr. care îi pointer în listă;

3. Write Invalidate Protocol:
→ fiec. bloc de dte. al mem. cache poate fi în una din stările:
single consistant = starea blocului care a fost citit din mem. princ. și adus

în cache-ul unui μP, el nu se mai află în

în cache-ul nici unui alt µP (single), și blocul nu a fost modif. din mom. aducerii din mem. princ. în mem. cache a µP (consistant);

multiple consistant = caract. (≠) copie a blocurilor din mem princ., blocuri citite în mai multe mem. cache ale mai multor µPs. și blocurile nu au suferit modif. → consistant;

single inconsistant = leg. de protoc. Write Invalidate → un bloc devine:

(a) dc. blocul fusese single consistant și µP. în a cărui cache se află blocul l-a modif. (s.n. single pt că e unic, inconsistant pt că între blocul citit din mem. princ. și blocul ce-l ≠ dif);

(b) blocul ar fi fost multiple consistant, µP în a cărui mem se află

(dc) blocul să-l fi modif. și spre deosebire de (a) s-a făcut invalidarea tuturor copiilor a. bloc în mem cache ale celorlalte µPs. care utilizează blocul respectiv.

invalid = starea în care trece un bloc în starea multiple dintr-o mem. cache ce a modif. conținutul dar care anunțase că blocul nu mai e consid.;

pg.50: Read Hit – single consistant
– single inconsistant
– multiple consistant
} citește și gata

Dc. blocul e invalid, read hit nu e posibil, pt că citește un bloc care ∄. Tentativa de a citi din bloc invalid = Read Miss.

Read Miss:

- Cz.1: blocul de care avem nevoie nu se găsește în nici o altă mem. cache ⇒ din mem principală se aduce blocul în cache-ul µP care a solicitat citirea și se setează starea blocului la single consistant;
- Cz.2: ∃ o sg. copie într-o mem. cache și această mem. e marcată cu starea single consistant. Atunci se citește blocul în cache-ul µP care a solicitat citirea și această copie, ca și deja ∃entă în cache-ul altui µP se marchează cu multiple consistant;
- Cz.3: ∃ mai multe ~~blocuri~~ copii ale blocului solicitat în mai multe cache-uri ale mai multor µP → se citește blocul în mem solicitant → se marchează doar ultima copie cu multiple

• Cz.4: ∄ o sg. copie marcată cu single inconsistant. Info. din blocul single inconsistant e fol. ptr. a actualiza info. mem. princ.
Ⓐ → single consistant.

Ⓑ Se citește blocul în forma actualizată → adus în mem. cache a µP solicitant și din single consistant → multiple consistant . . . . ?

Write Hit
- single consistant —dev.→ single inconsistant
- ~~—u—~~ inconsistant → și mai ~~—v—~~
- multiple consistant → blocul modificat devine single inconsistant → comandă de invalidare a tuturor copiilor

Write Miss

(1) → ∄ nici o copie nicăieri ⇒ iau din mem. principală și pun în cache-ul solicitant → single inconsistant;

(2) → ∄ o sg. copie single consistant ⇒ aduc copia în cache-ul µP care face modif. → marchez copia cu single inconsistant și o invalidez pe cea care era single consistant;

(3) → m. multe copii multiple consistant ⇒ se aduce o copie în cache-ul solicitant, se marchează cu single ~~consistant~~ inconsistant și se invalidează toate celelalte copii ale blocului;

(4) → ∄ o sg. copie single inconsistant și se actualizează mem. principală, aduc dte. în cache-ul solicitant.