Resum AC

Cache bàsica

- Localidad temporal: si accedim a una pos és molt probable que hi tornem a accedir

- Localitat espaial: Si accedim a una pos és molt porbable que accedim a una propera

Qalsevol memòria cache te:

- Algoritme d'emplaçament: a quines línies de la MC es col·loca un bloc (associativa, directe, conjunts)

- Algoritme de reemplaç: aleatori, FIFO (first in first out), LRU (least recently used)

- Politica d'escriptura: write through(s'actualitza MC i MP, el temps de servei és Ta a MP), copy back(s'actualitza MC, dirty bit, s'actualitza MP quan es canvia la línia), Write allocate (si falla es porta el bloc de MC a MP), write no allocate (el bloc no es porta a MC si falla)

Medir rendiment:

Texe = N · CPI · Tc

CPI = CPI ideal + CPI mem

CPI mem = nr · (Tma - tsa) = nr(num de referencies) · m · tpf

Tsa = temps d'acces

Memòria Virtual

Esquemes bàsics de traducció de direccions de espai logic(generades pel processador) a direccions d'espai físic(amb les que s'accedeix a memòria):

- **Segmentació**: separar el programa en segments

- **Paginació**: l'espai lògic es divideix en blocs de tamany fixe. La MP es divideix en marcs del tamany d'una pàgina. Els programes es divideixen en pagines. Les pàgines es copien de disc a MP quan son referenciades. Cada procés té la seva taula de pàgines.

Desplaçament = log2(tam pàgina)

VPN + desplaçament = @lògica

PPN + desplaçament = @física

Necessitem una entrada a la taula per cada VPN i a cada entrada necessitem PPN bits, per tant → tam min taula = 2^(VPN)·PPN

Com que la taula de pàgines seria molt llarga, tenim la **TLB** que és una cache de la taula. És completament associativa. Té bit de validesa. Copy back + WA

Cache avançada

Texe = N · (CPI ideal + CPI mem) · Tc

Els fallos de cache es poden dividir en 3 categories: Carga(la primera vegada que s'accedeix a una pos), capacitat(totes les línies que necessita un programa no hi caben), conflicte(quan més d'un bloc es mapeja al mateix lloc de la MC).

Tècniques bàsiques per millorar rendiment cache:

- Augmentar tam bloc: Menys fallos de carga

- Augmentar tam cache: menys fallos de capacitat i conflicte, augmenta temps accés i consum

- Augmentar grau associativitat: meys fallos de conflicte, augmenta temps accés

- Cache multinivell: una cache petita amb molts miss i tsa baix i una gran al reves

- Donar prioritat a les lectures davant les escriptures: baixa el CPI wr

Tècniques avançades per millorar rendiment cache:

- Reduir cost d'un encert en cache: cache petita i simple, predicció de via i trace caches

- Augment d'ample de banda de cache: cache segmentada i cache multibanc i no bloquejants.

- Reduir cost de fallo: early restart i merging write buffers

- Reduir tasa de fallos: optimitzacions del compilador

- Reduir cost i tasa de fallos via paral·lelisme: pre busqueda hard i prebusqueda soft

OPTIMITZACIONS:

Ta = temps d'accés

- Cache simple per reduir Ta: temps d'accés depèn del camí crític

- Cache petita per reduir Ta: el temps d'accés depèn del tamany de la cache

- Predicció de via per reduir Ta: Els programes tenen una execució predible(localitat) i s'actualitza la taula de predicció amb el comportament dels accessos previs.

- Trace cache per reduir Ta: es guarden seqüències dinàmiques d'execució d'instruccions

- Cache segmentada per augmentar ample de banda: la latència d'1 accés augmenta però l'ample de banda augmenta

- Non blocking cache: en cas de miss el procés només es para quan necessita la dada

- Cache multibanc: redueix consum, pots realitar accessos concurrents.

- Reduir penalització per miss: **early start:** quan es carrega la dada a MC que ha fet miss s'envia a la CPU. **Transferència en desordre i continuació anticipada:** s'envia primer la dada que ha fet miss.

- Buffers d'escriptura: llegir el bloc que es porta a MC té prioritat abans que escriure l'anterior bloc a MP. Per això es fa servir un buffer on es van guardant els blocs bruts. Útil amb les caches WT.

- Optimitzacions de codi per reduir fallos: reordenar codi, reordenar dades, fusió o intercanvi de bucles.

- Prefetch d'instruccions i dades per reduir tasa de fallos i penalització: a vegades es poden portar instruccions inútils. Ho fa el compilador o el programador low level.

RAM

Tipus de memòria:

- **Memòria estàtica**: cache, cada cel·la és un biestable.

- **Memòria dinàmica:** MP, cada cel·la es comporta com un condensador. Lentes, baix consum.

Lectura DRAM:

1. Decodificar @fila, s'activa senyal wordline

2. S'accedeix a totes celes de la fila i les seves dades s'enien als amplificadors de senyal i es recupera la tensió

3. Decodificar @columna, es selecciona una bitline i s'envia la dada al buffer R/W

4. S'envia la dada al exterior des del buffer R/W

5. La lectura es destructiva, s'ha de tornar a escriure tota la fila per recuperar els valors originals i precargar els bitlines per el següent accés a memòria

Escriptura DRAM:

El 3) i 4) igual amb la única diferència que el es reescriu amb la dada que entra al buffer R/W.

Valors:

- Temps d'accés (Trac): retard des que es subministra direcció fins que s'obté dada → latència de memòria.

- Temps de cicle(Trc): interval de temps mínim entre dos accessos → ample de banda

- Temps d'accés a columna(Tcac): retard màxim des de que se subministra la direcció de columna fins que s'obté dada

<u>Evaluació i optimitzacions:</u>

Tots els accessos a MP son per R/W cache. Normalment es necessiten 4 accessos a memòria.

Temps llegir línia cache = Trc * 4

Ampla de banda = Tam línia cache / temps llegir línia cache

- FPM DRAM: Possible optimització: les dades d'una caché son consecutives a MP, com estan a la mateixa fila només cal canviar @col. (localitat espacial). En aquest cas el temps cost de llegir una línia de cache:

Temps total = Trac + Tcac * 3

Ampla de banda pic = tam al que s'accedeix (8B) / Tcac

- EDO DRAM: S'afegeix un registre perquè es pugui solapar l'enviament de la nova @col amb l'accés a dades. Com que la dada està en un registre, podem enviar @col abans d'acabar l'enviament de la dada. Ara ha millorat la latència de columna → millora temps total i ampla de banda

- BEDO DRAM: S'afegeix un contador per calcular @col +1, +2 i +3 a partir de @col. Millora latència de columna.

- SDRAM: S'utilitza una memòria síncrona pk és més ràpida, no té soroll, es pot augmentar Freq de funcionament. Temps total = Temps accés + temps transferència * 4

- DDR SDRAM: Si enviem una dada quan puja i una quan baixa el rellotge, millorem substancialment l'ample de banda.

Comandes de les SDRAM:

Active(ACT), READ (RD), WRITE (WR), PRECHARGE(PRE)

RAM Avançada

Les DRAMs estan dividides en bancs. Això permet fer accessos concurrents a diferents bancs, amagar precarrega, amagar refresc, tenir arrays de memòria més petites →menys temps d'accés i consum d'energia. La MP utilitza DIMMs. Un DIMM te un num determinat de chips. Un chip té uns 8 bancs. Cada banc te x files i col.

Refresh: com la DRAM son condensadors tenen corrent de fuga. Per no perdre les dades s'ha de refrescar les files, fila a fila. El refresc es fa amb un ACT i després un PRE. Periode refresc = 64ms.

Sistemes d'emmagatzematge

El controlador de disc produeix una interrupció cada vegada que te una dada nova. Això interromp la CPU. Per això tenim un controlador d'accés directe a memòria (DMA). Si la cpu vol fer una operació de disc ha de programar el disc i el DMA(R/W, num dades, on deixar-los).

Un cop transferides totes les dades, la CPU es sincronitza amb el disc (amb interrupció).? Només s'utilitza DMA per transferir bloques de datos.

Robo de ciclo: CPU té preferència sobre accés a MP. Si KDMA vol accedir a MP, envia BR a CPU. Si CPU no l'està utilitzant, li respon enviant BG. Quan KDMA acaba, descativa BR i CPU desactiva BG.

Transferència a ràfega: Es transfereix més d'una dada i es manté BG i BR activat de mentres.

Que la CPU i la KDMA puguin accedir a la MP genera problemes:

- De coherència amb la MC: carregar a disc bloc modificat a MC però no a MP(solució: Write Through, KDMA accedeix només a zones no cacheables, buidar MC cada vegada que es crida KDMA). El mateix problema però ara KDMA escriu a MP i la lectura de MC dona un valor incorrecte(solució: KDMA a zones no cacheables, buidar MC cada vegada que e crida KDMA).

- Traducció de direccions: el SO per poder accedir a KDMA ha de anar a la taula de pàgines per trobar la direcció física. ?

DISC DUR

Gran, no volàtil, tecnologia magnètica, element mecànic, organitzat en cares, pistes, sectors. Mètodes d'accés:

- Modo MHS : S'accedeix a disc amb especificacions geomètriques(cilindre, cara i sector).

- Modo LBA: els sectors estan numerats de 0 a n-1. Es necessita un mecanisme de traducció.

Paràmetres de rendiment Disc dur:

- Average seek time: cost mig de posar el capçal en el cilindre que es vol accedir (en ms).

- Latency: cost mig de situar el capçal al sector que es vol accedir (en ms). Depen de RPM

- RPM: en mode normal el disc sempre està girant.

- Average access time: Average seek time + latency.

- Transfer rate: MB/s

- Cache: es guarda temporalment info llegida del disc (o pendent d'escriure). Possibles estratègies de prefetch.

- MTTF: temps mig entre fallos

- Interfície: comunicació entre disc dur i pc.

Fiabilitat: els fabricants ho fan amb el MTTF. És més útil el AFR(annual failure rate):

Discs que fallen en un any = (num discs · hores/disc)/MTTF → % = AFR

Sector: unitat mínima d'acces a disc. (GAP – SYNC - @ - DATA – CRC – ECC - FP)

- Gap: marca física que indica la separació entre sectors
- SYNC: 10 bytes que fiuen freq i amplitud amb les que s'ha gravat la info
- @ (address mark): id de sector + informació d'estat
- DATA: 512 bytes de dades codificades en RLL
- CRC: checksum que comproba integritat del bloc de info
- ECC: informació redundant per detectar i corregir. El firmware del disc dur s'encarrega de gestionar errors. El SO només rep el bloc que ha sol·licitat o un error. Com més bits redundats més robust → necessita un controlador més car
- FP: informació interna per sincronitzar

S.M.A.R.T: tecnologia que monitoritza els components mecànics i electrònics del disc per avisar o corregir errors. Quan hi ha un "avis de S.M.A.R.T" és molt probable que falli.

Codificació: Per realitzar lectures i escriptures segures es poden fer un màxim de transaccions en un cert temps.

SSD: memòria no volàtil. Utilitza NAND Flash(accés aleatori a blocs i seqüencial dins d'aquests, num limitat d'escriptures i borrades, menor fiabilitat que NOR Flash i utilitzen ECC per augmentar-la).

RAID: esquema estandarditzat de múltiples discs. Molts discs operant en paral·lel per incrementar ample de banda. Protegeix les dades contra la fallada d'un disc. Es perd una part de la capacitat dels discs. Els sistemes RAID professionals tenen alimentació i ventiladors duplicats per si falla un. Hi ha de RAID 0 – RAID 6, amb diferents toleràncies a fallos, rendiment i cost. MTTF de n discs = MTTF / n

El SO veu el RAID (un conjunt d'unitats físiques) com una única unitat lògica. Té tècniques d'entrellaçament:

- RAID 0: DISK Striping: Divideix els arxius en stripes i es separen entre els diferents discs. No hi ha protecció de les dades. Accés seqüencial a arxius de gran tamany.

- **RAID 1: Mirroring:** Es dupliquen les dades en discs addicionals. Les dades es poden llegir de qualsevol de les copies. Les escriptures són és lentes perquè s'ha de fer als dos llocs. És car

- **RAID 2 : Redundancia a través del codi Hamming:** Fa entrellaçament a nivell de bit. L'operació d' E/S accedeix al mateix sector de tots els discos en paral·lel. Fa servir ECC. Pot detectar i corregir 1 disc o detectar 2 discs que fallen.

- **RAID 3: Accés síncron amb un disc dedicat a la paritat:** entrellaçament a nivell de byte (bit a HP). Dedica un disc per guardar info de paritat. Amb ECC detecten errors de disc i per recuperar dades es calcula la XOR de la info dels altres discs. E/S accedeix al mateix sector de tots els discs. El rendiment de transacció és pobre.

- **RAID 4: accés independent amb disc de paritat:** entrellaçament a nivell de tira. Es pot accedir als discs de forma individual. Només té un disc per paritat i si falla no es pot recuperar. Les escriptures son costoses ja que sempre s'escriu al disc de paritat.

- **RAID 5: accés independent amb paritat distribuïda:** els blocs de paritat estan distribuïts entre tots els discs. Evita el coll d'ampolla de RAID4(disc de paritat). S'accedeix als discs en paral·lel i de forma independent. Una escriptura requereix 2 lectures i 2 escriptures.

- **RAID 6: accés independent amb doble paritat:** com el RAID 5 però té un segon esquema de redundància distribuït pels discos, per tant té una tolerància molt alta. Pot recuperar info encara que fallin 2 discs. Escriptures costoses (3 lectures i 3 escriptures).

Fiabilitat RAIDs: temps entre fallo de 1 disc → $p = 1 - e^{-\lambda t}$ on:

p = probabilitat que es produeixi un fallo, λ = 1/MTTF, t = temps transcorregut

En un RAID 0 si falla un disc el sistema falla. En els 1, 3, 4, i 5 si falla un disc el sistema segueix operatiu. Però si durant el MTTR falla un altre disc falla el sistema. En un RAID 6 el sistema falla si falla un 3r disc.

$$MTTF_{RAID5} = MTTF_N \times MTTF_{N-1} / MTTR = MTTF_{disco}^2/N \times (N-1) \times MTTR$$

Disseny del joc d'instruccions

Tipus d'arquitectures en funció dels operands explícits:

- **Màquina de pila:** sense operands explícits amb excepció de op de salt i accés a memòria.

- **Màquina d'acumulador:** te un acumulador (un registre) com a operand implícit a totes les op.

- **Arquitectura GPR:** accés ràpid a registres. Totes els operands son explícits.

- **Màquina de Registre/Memòria:** dos operands explícits, un d'ells pot estar a memòria.

- **Màquina de Memòria/Memòria**: 3 operands explícits, poden estar a memòria.

- **Màquina Load/Store**: 3 operands explícits en memòria. Necessiten op load i store.

Direccionament: algoritme utilitzat per accedir als operands d'una instrucció. Modes diferents:

- **Mode registre:** l'operand està aun registre de CPU. Ràpid però es necessita movl

- **Mode immediat:** L'operand està en la instrucció.

- **Mode Absolut:** La direcció de l'operand està a l'instrucció.

- **Mode registre indirecte:** la direcció codifica registre que conté la direcció del operand.

- **Mode base + desplaçamet:** La direcció es calcula amb un registre i un desplaçament.

- **Mode indexat:** La direcció s'obté sumant dos registres.

- **Mode escalat**: útil per a vectors. (%ecx, %ebx, 4)

Modes de direccionalment combinats:

- **Post-indirecció:** la dada obtinguda amb els modes anteriors és l'operand.

- **Post-escalat:** el VAX11 permet combinar escalat amb qualsevol altre.

CISC: instruccions molt complexes. Poca memòria. µprograma: augmenta la densitat de codi i redueix el tràfic a memòria i mida del programa. Molts modes de direccionament. Instruccions de longitud variable(dificulta fetch i decode). La µmemòria es comença a fer massa gran.

RISC: instruccions molt simples. Els compiladors generen codi més eficient. Molts registres de propòsit general. Operands, variables locals i paràmetres en registres. S'accedeix a memòria amb load i Store. Execució segmentada → 1 instrucció/cicle

µprogramació: tècnica utilitzada per simplificar disseny de la UC (unitat de control) dels CISC. El µprograma és l'intèrpret de LM. El contingut de µmemòria és el µprograma, fet per µinstruccions que equivalen a estats del sistema seqüencial de la UC. µinstruccions es codifiquen amb:

- **control:** Bits que governen els circuits de la UP.

- **dir:** direcció de la següent µinstrucció en cas que s'hagi de fer un sal.

- **cond:** forma d'evaluar els flags els flags i bits de IR per si saltar a dir

En un µprograma s'identifiquen les fases en que es poden dividir l'execució d'una instrucció de LM: Fetch, decodificació i execució detallada.

Segmentació i paral·lelisme en el disseny de computadors

Paral·lelisme a nivell d'instruccions:

Processadors segmentats:

Per simplificar segmentació es fa servir LM tipus RISC. Per simplificar hardware es busca que totes les instruccions utilitzin la mateixa segmentació (F, D/L, A, M, W). La segmentació té límits: risc de dades (la següent instr llegeix dada que encara no s'ha escrit), risc de control (si es salta en una instr, ja s'haurà iniciat l'execució de dues instruccions), risc estructural (un únic recurs es vol utilitzar per 2 instr)

Processadors superescalars:

Permet iniciar més d'una operació per cicle. Les instruccions poden tenir temps d'execució diferents. OoO (out of order processors). Les instruccions es llegeixen en ordre però poden executar-se en desordre. Si els seus operands no estan disponibles les instr es bloquegen. Quan els operands estan disponibles i la UF està disponible s'executen.

VLIW: Very Long Instruction Word.

Una instrucció especifica múltiples op diferents. El compilador decideix quan i on s'executa la op, a diferència dels superescalars que ho decideix el hardware.

Paral·lelisme a nivell de dades:

SIMD: Single Instruction Multiple Data. Una instrucció que permet operar amb múltiples dades del mateix tipus. Aplicacions multimèdia.

Processadors vectorials: orientats a càlculs científics en coma flotant. Instruccions que operen amb vectors de números en coma flotant. Permet un alt grau de segmentació en les UF i tenir temps de cicle molt petits. Molt eficient accedint a dades. Les dades que utilitzen les UF no passen per MC.

Paral·lelisme a nivell de thread:

Multithreading: Quan un thread està aturat, que un altre thread utilitzi les UF.

Multiprocesadores: es fa per aconseguir més velocitat o per poder executar més aplicacions per unitat de temps.

Poden tenir una memòria compartida (un únic espai de direccions compartit)o distribuïda(només accedeixen a la seva memòria local).

Poden utilitzar el model de variables compartides(comparteixen dades) o model de pas de missatges (les dades no locals s'accedeixen utilitzant missatges entre processos). S'ha d'utilitzar MC per a cada procés de gran capacitat. Això pot generar errors de coherència de memòria.

Les xarxes d'interconnexió poden ser un bus en comú, bussos múltiples o crossbar. Aquestes xarxes també son essencials en els multiprocessadors amb memòria distribuïda.