A memória működése szoftverfejlesztői szemmel

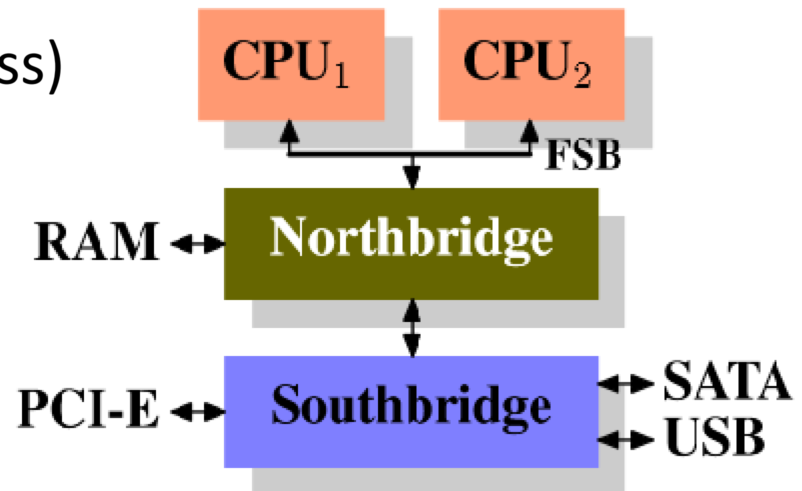
Smidla József
Operációkutatási Laboratórium
2013. november 20.

Tartalom

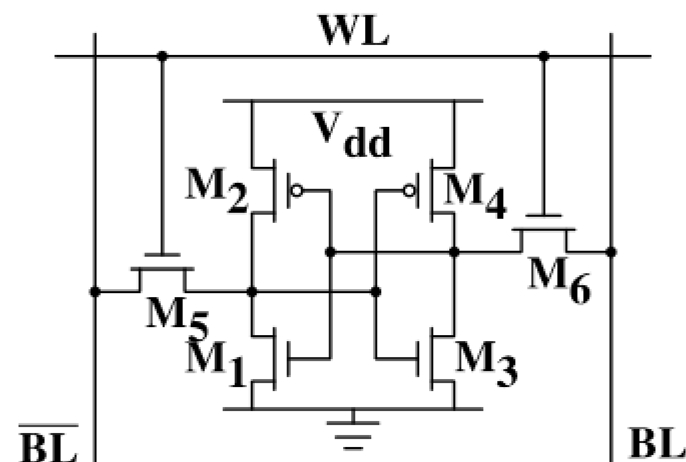
- A memória felépítése
- Cache memória
- Mérési adatok
- Cache detektáló program

Architektúra

- FSB: Front Side Bus, az északi hidat köti össze a processzorokkal
- Északi híd: Memóriavezérlő, összeköti az FSB-t a memóriával, és néhány perifériával
- Szűk keresztmetszet csökkentése:
  DMA (Direct Memory Access)
- Északi híd és memória közötti szűk keresztmetszett csökkent-hető több csatornával (DDR2-nél 2 csatorna)



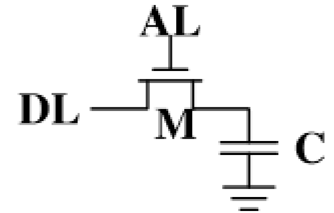
Memória fajták



- SRAM: Static RAM
- 0 és 1 állapotok egyikét tárolja
- 1 db cella 6 tranzisztort igényel
  (4 tranzisztorral is megoldható,
  de van hátránya)
- Folyamatos áram ellátást igényel
- Ha a WL vonalat felhúzzuk, a kimeneten egyből megjelenik a tartalom, nem kell várakozni
- Stabil állapot, nem kell frissíteni
- Drága

Memória fajták

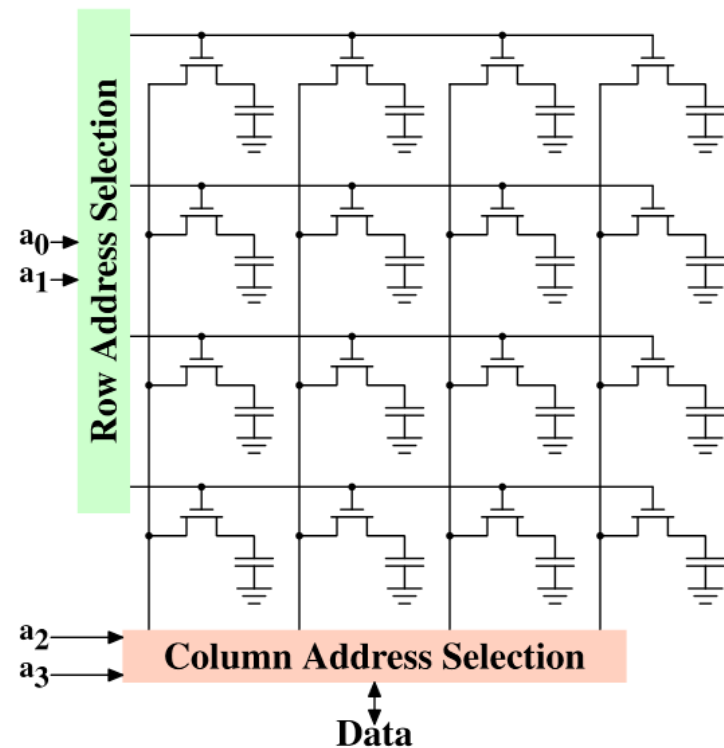
- DRAM: Dynamic RAM
- 1 tranzisztor és 1 kondenzátor
- Ha az AL vonalat bekapcsoljuk, akkor a DL felé a C kondenzátor töltöttségétől függően töltés áramlik, vagy nem



- Probléma: az olvasás kiüríti a kondenzátort, ezért vissza kell tölteni a tartalmát
- Idővel a kondenzátor tartalma szivárog
- A legtöbb DRAM chipben a cellákat 64 ms-ként frissíteni kell
- Lassabb, mint az SRAM
- De legalább olcsó

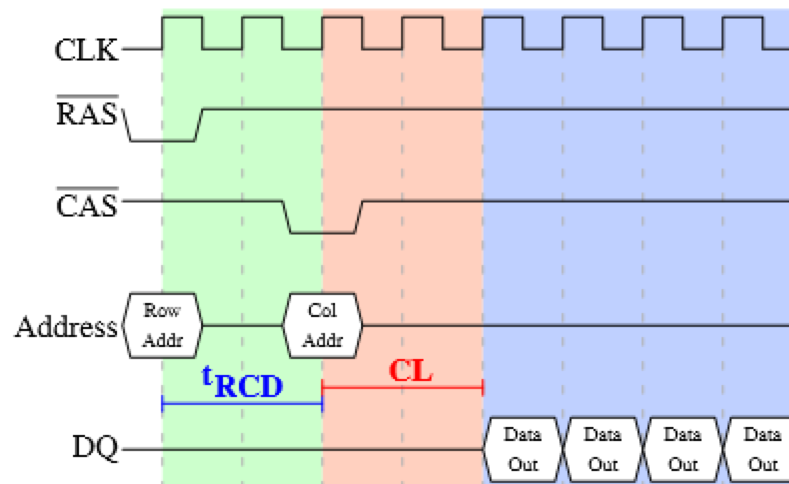
DRAM elérés

- Minden cellának egyedi címe van
- A címet elfelezzük, az egyik felével az oszlopot, a másikkal a sort címezzük meg (különben egy túl nagy demultiplexerre lenne szükség)



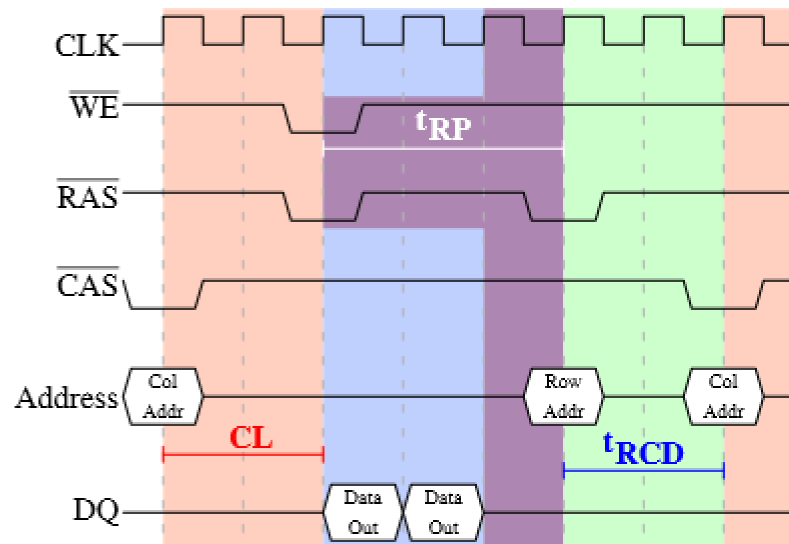
DRAM elérés

- Először a címvonalra rakjuk a sor címet, közben az RAS vonal aktív
- Majd t_{RCD} idő elteltével a címvonalra rakjuk az oszlop címet is, ezt a CAS vonallal jelezzük
- Végül CL idő elteltével megjelenik az adat a DQ vonalon



DRAM elérés

- A következő eléréshez a RAS vonalnak újra kell töltődnie, ehhez t_{RP} idő kell
- A t_{RP} 1 ciklussal tovább tart, mint az átvitel
- A példában 7 ciklusból 2 ciklusban van adat átvitel

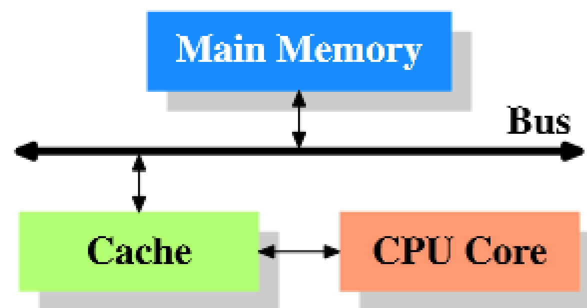


Cache

- A fő memória DRAM-ból áll
- A mai számítógépeken nem ugyanolyan magas órajelen működik a CPU és a memória (nagyon nehéz, és költséges a memóriát is ilyen sebességgel működtetni)
- A memória modul távol van a CPU-tól, a vezeték is sokat késleltet
- A DRAM lassabb, mint az SRAM

Cache

- A CPU és a memória közé illesszünk cache-t
- A cache-ben tároljuk a gyakran használt adatokat
- A cache tartalmát hamarabb kiolvashatjuk

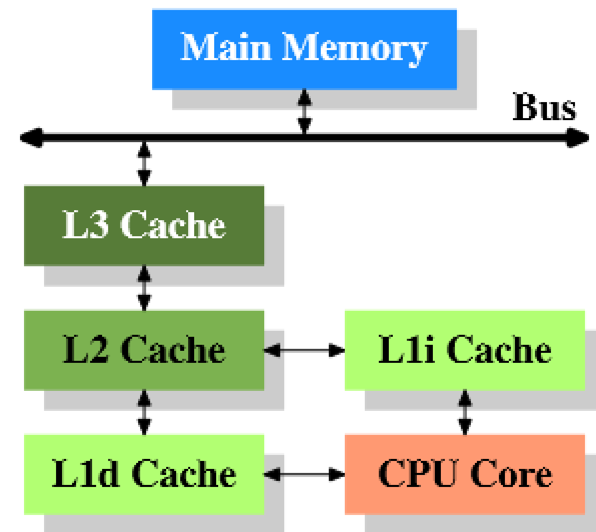


Cache szintek

- Manapság több szintű cache-t alkalmaznak
- Minél magasabb a cache szintje, annál messzebb van a CPU-tól
- Az L1 cache szétválik utasítás és adat cache-re

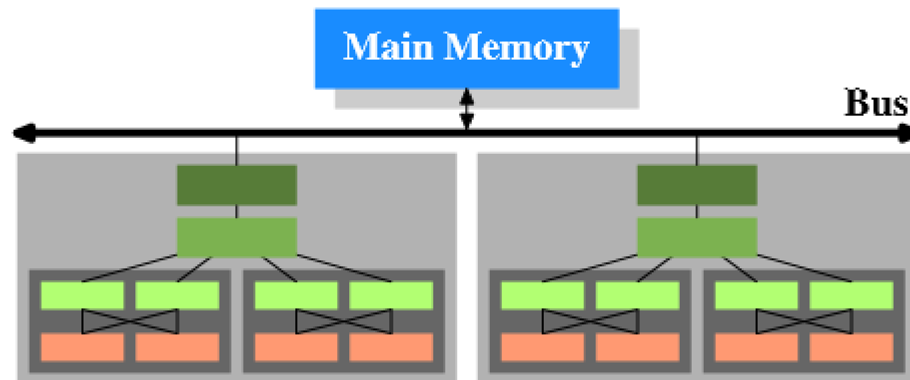
Cél	Órajel ciklusok
Regiszterek	≤ 1
L1d cache	~ 3
L2 cache	~ 14
Memória	~ 240

Pentium M processzor



Cache megosztás

- Több processzor esetén minden CPU külön cache-el rendelkezik
- Egy CPU-n belül minden mag saját L1 cache-el rendelkezik
- A CPU magasabb szintű cache-ein osztoznak a magok



A cache használata

- A CPU-nak szüksége van egy bizonyos memóriacímen lévő adatra
- A CPU megnézi, hogy ez az adat szerepel-e a cache-ben, ha nem, akkor betölti azt
- Ha 4 bájtra van szükségünk, akkor nem csak azt a 4 bájtot töltjük be, hanem 32, vagy 64 bájtot, azaz egy teljes cache line-t
- A cache line beolvasása hatékonyan megoldható külön RAS és CAS jelek nélkül

A cache használata

- A kiolvasott információ az L1 cache-be töltődik be
- Ha a cache line 64 bájt, akkor a cache line-on belül minden bájtot 6 bittel címezhetjük meg (offset)
- A memóriacím többi részét arra használjuk, hogy a bájtot megkeressük a cache-en belül
- Ha írunk egy adott címre, akkor a hozzá tartozó cache line-t be kell tölteni a cache-be
- Részleges cache line nem tárolható a cache-ben

A cache használata

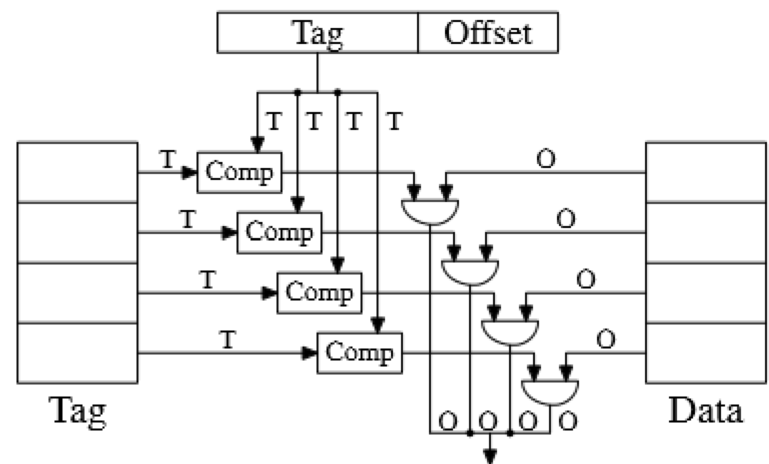
- Mi történik, ha nincs elég hely az új adatnak az L1 cache-ben?
- A CPU kiválaszt egy cache line-t, és azt az L2-be, ha ott sincs hely, akkor az L3-ba, végül a fő memóriába dobja
- Exlusive cache: minden adat legfeljebb csak 1 cache-ben szerepel (AMD, VIA processzorok)
  - A cache line betöltése gyors
- Inclusive cache: Az alacsonyabb szintű cache-ek tartalma részhalmaza a magasabb szintűnek (Intel)
  - A cache line kidobása gyors

Cache koherencia

- Ha egy CPU írja a memóriát, akkor a megfelelő cache line-t megjelöli (dirty, nem ugyanaz van a cache-ben, mint a memóriában, majd valamikor visszakerül a tartalom a memóriába)
- Ha több CPU van, akkor a memóriát ugyanolyannak kell látniuk: cache koherencia
- Cache koherencia protokollok, például MESI
- Alapelv: A dirty cache line csak 1 cache-ben létezhet, a tiszta másolatok (clean copy) akármennyiben

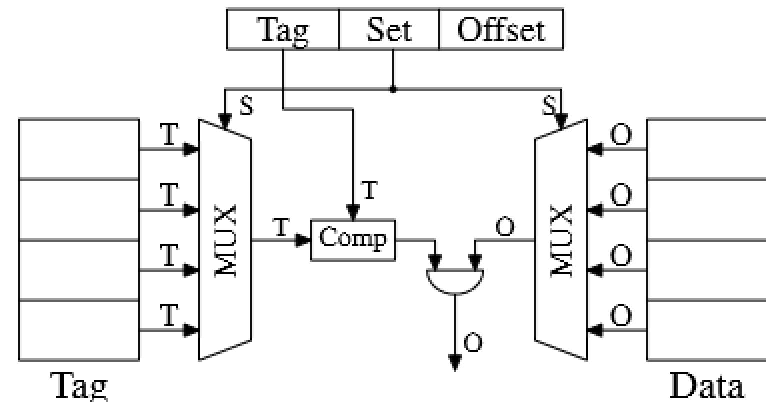
Full associative cache

- A memóriacímet egy tag és egy offset részre osztjuk
- Minden cache line tag-jét tároljuk
- Egy memóriacím tag-ét párhuzamosan összehasonlítunk minden eltárolt tag-el
- Gyors
- Technikailag nagyon nehéz egyszerre sok komparátort működtetni
- A mérete limitált, csak speciális cache-ekben alkalmazzuk (TLB)



Direct-Mapped cache

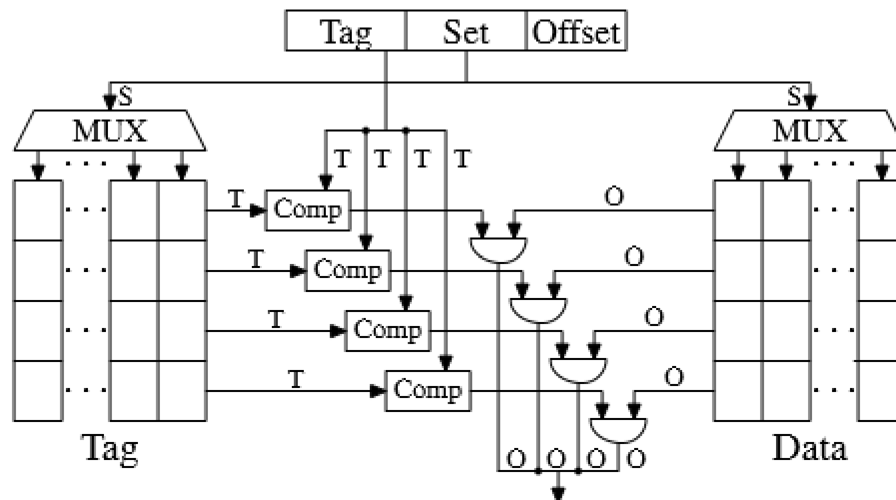
- A memória címet már 3 részre osztjuk
- A set adattaggal címezzük meg a cache-ben a cache line-okat
- A tag-et csak ellenőrzésre használjuk, hogy a különböző, de azonos set-el rendelkező címeket megkülönböztessük
- Előny: nagyobb lehet, mint a full associative cache
- Hátrány: Bizonyos, azonos távolságra lévő memóriacímeket nem tárolhatunk egyszerre



Set-Associative cache

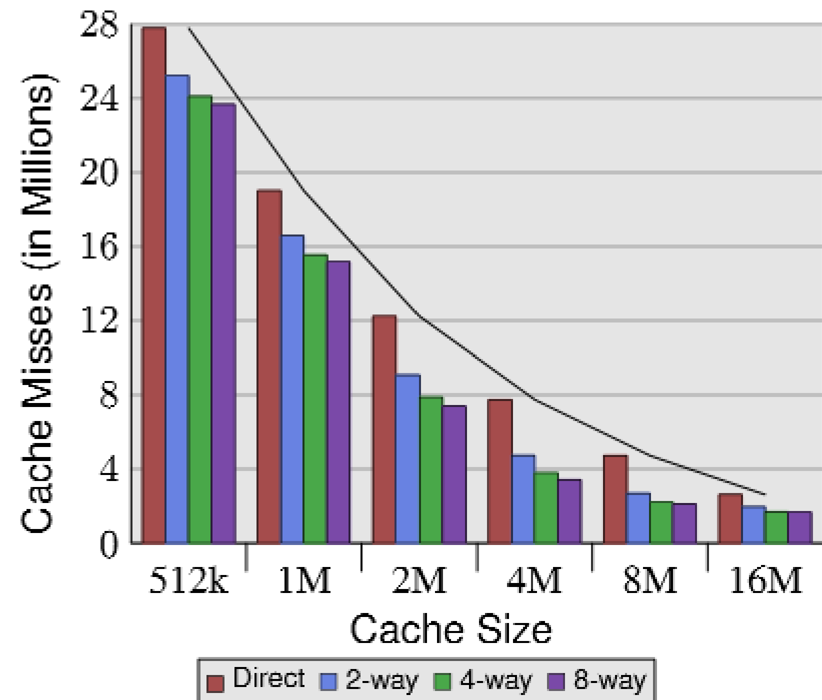
- A direct-mapped cache-t egészítjük ki úgy, hogy több azonos set-ű memóriacímet is tárolhatunk
- Többnyire a set-ek számát növelik
- Cache méret:

asszociativitás * halmazok száma * cache line mérete



Set-Associative cache

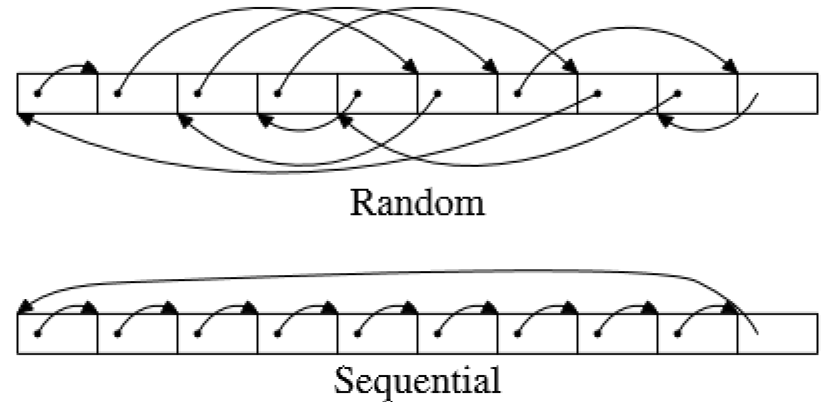
- Cache miss: a keresett adatot nem találjuk meg a cache-ben
- Cache méret és asszociativitás hatására a cache miss arányra:



Teszt

Tömb bejárás

```
struct l {
   struct l * n;
   long int pad[NPAD];
};
```

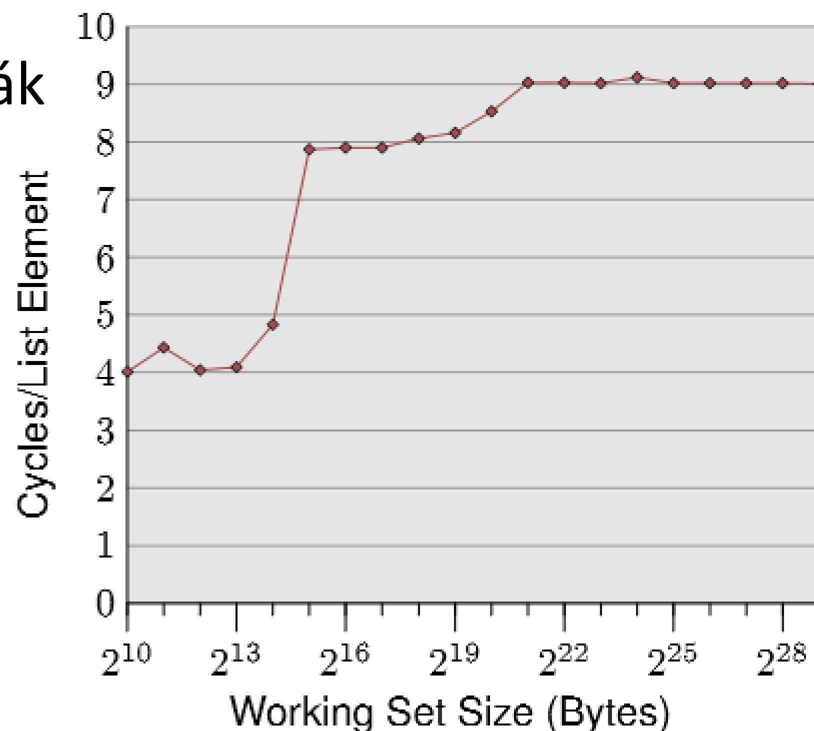


Az NPAD-al állíthatjuk, hogy mekkora legyen az n-ek közti távolság

16 kB L1d, 1 MB L2 cache

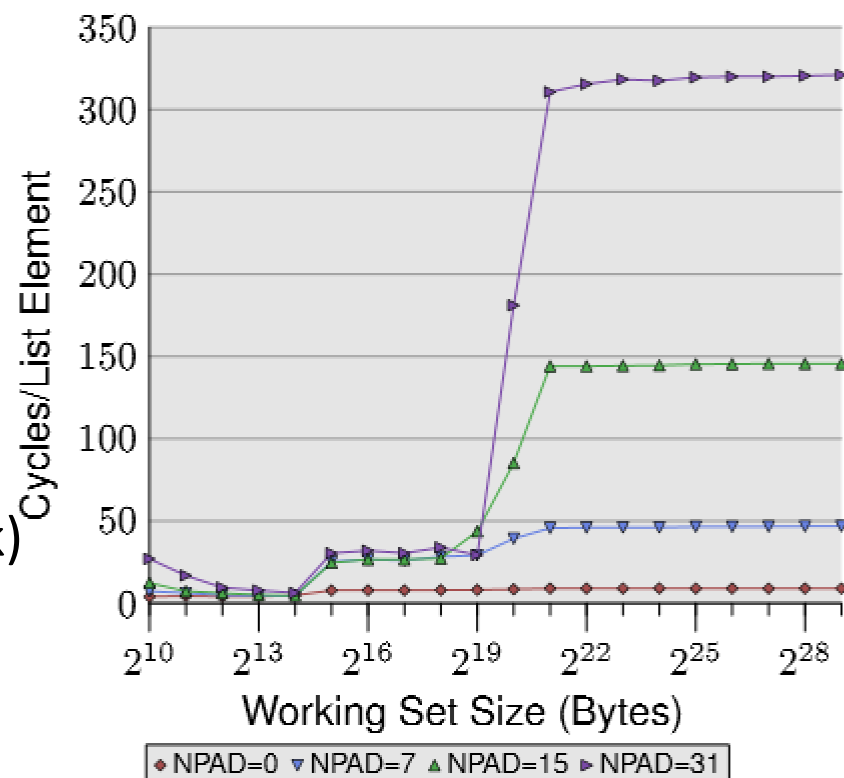
Szekvenciális bejárás

- NPAD = 0
- Az első két mérés zajjal terhelt
- A határok nem élesek, mert a rendszer más elemei is használják a cache-t
- A határok a cache méreteknél vannak:
  - 2^{14} bájtig: L1d
  - 2^{15}-től 2^{20} bájtig: L2
  - 2^{21} bájttól: memória is
- Magyarázat: hatékony prefetch



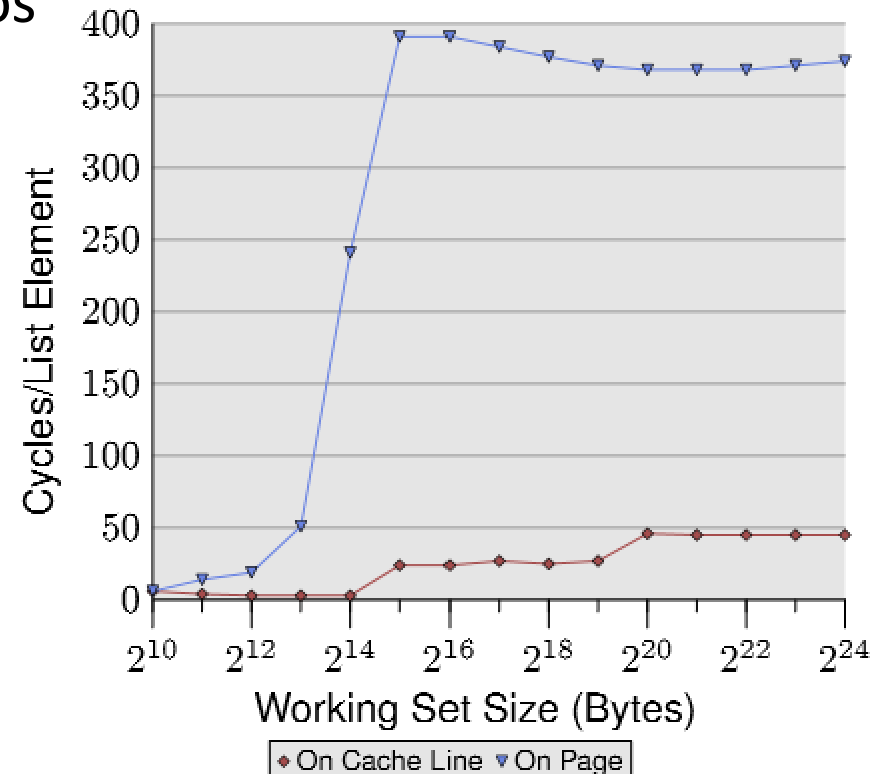
Szekvenciális bejárás

- NPAD = 0, 7, 15, 31
- Az n elemek egyre messzebb kerülnek egymástól
- Az L1d használatakor nem kell prefetch, a vonalak illeszkednek egymásra
- NPAD = 7-től az L2 használatakor nem használható a prefetch, az L2 elérési ideje jelenik meg
- Az L2-n túllépve tovább romlik a helyzet, megint nem használható a prefetch (kis adatokat használunk)
- A TLB cache is közrejátszik a lassulásan



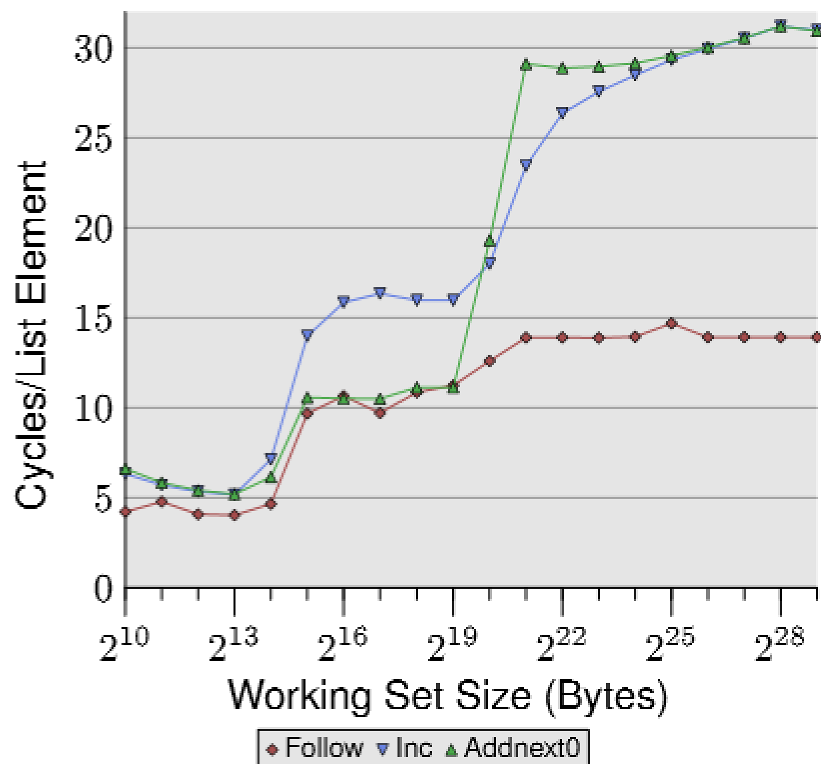
A TLB hatása

- TLB: a virtuális címet fizikai címmé kell alakítani, a TLB ezt gyorsítja fel
- Két mérés: a cache line-ok azonos lapokon, illetve külön lapokon
- Nagy ugrás 2^{13} bájtnál
- 64 bejegyzés van a TLB-ben
- Ha a TLB nem használható ki, akkor a fizikai cím előállítása hosszadalmasabb, ezért nő a címzési idő



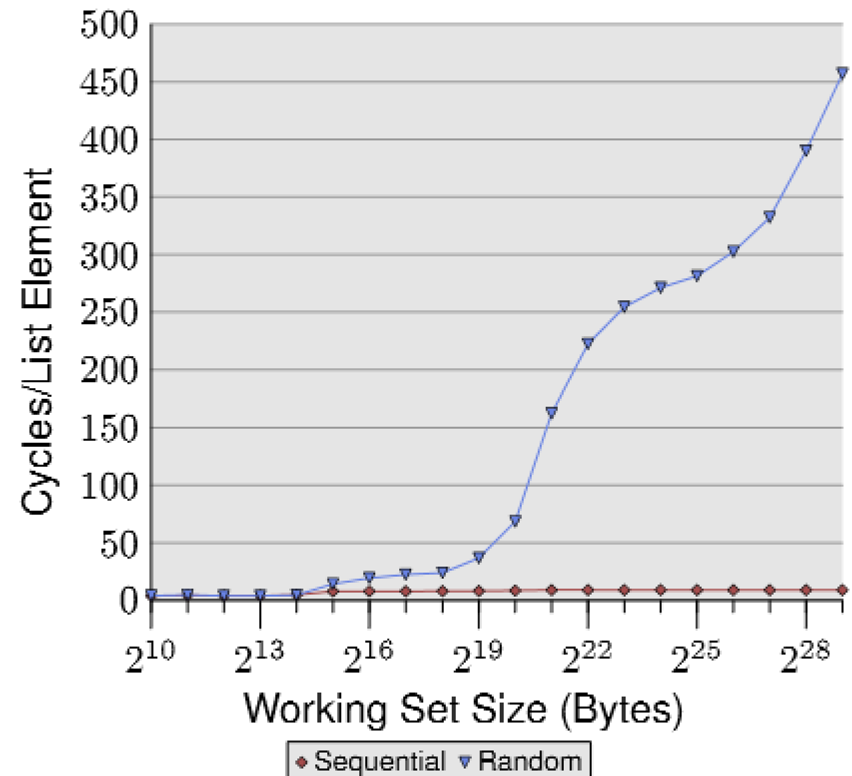
Írási sebesség

- Follow: az eddigi teszt (NPAD most mindig 1)
- Addnext0: a következő elem hozzáadása a mostanihoz
- Inc: pad[0] inkrementálása
- Várakozás: Az Addnext0 lassabb, mert több memória műveletet igényel
- Azonban: Amíg az L2-ben elfér minden, addig az Addnext0 szoftveresen prefetch-eli a követ-kező adatot!
- Az L2-t átlépve az Inc és Addnext0 hatására a módosított tartalmat vissza kell írni a memóriába, mivel gyakran kell új cache line-nak helyet csinálni



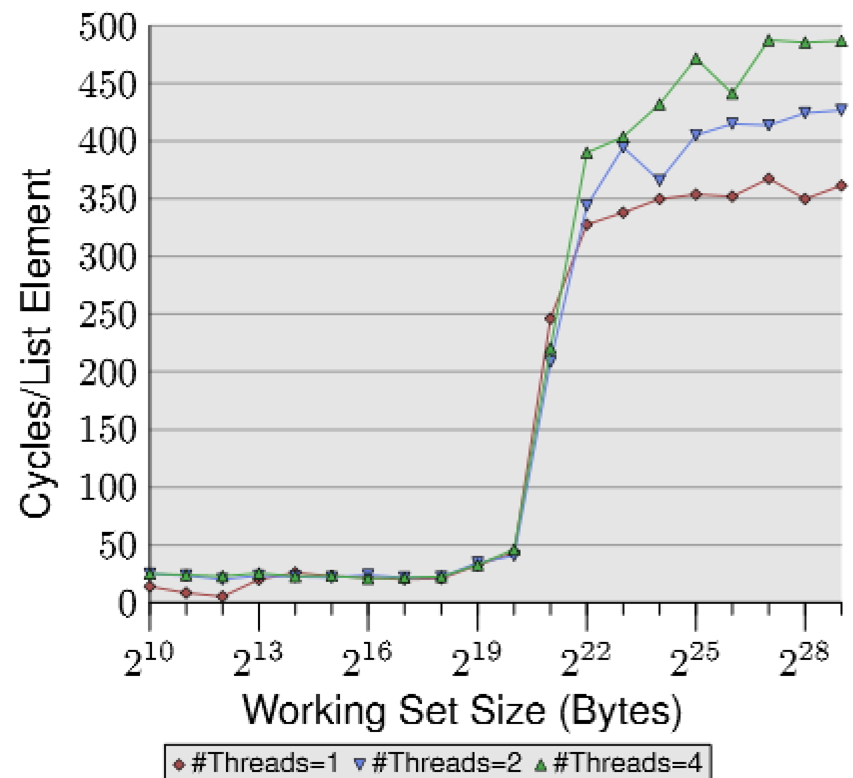
Véletlenszerű elérés

- Az elemeket véletlenszerű sorrendben járjuk be
- Nincs prefetch
- A TLB hatása is érződik



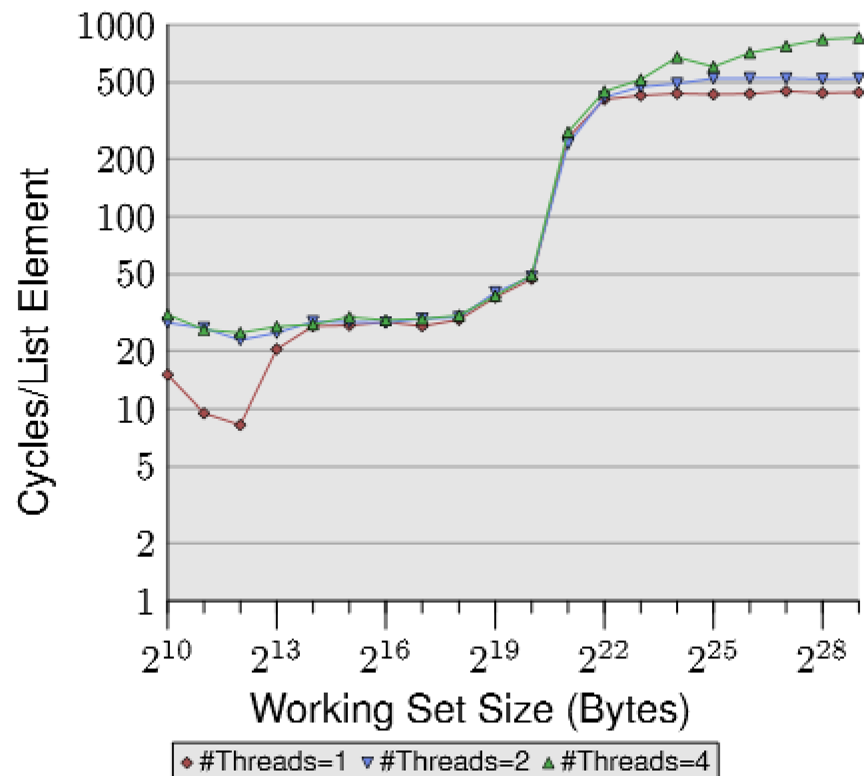
Több szálú elérés

- Ha túlléptük az L2 cache méretét, akkor több szál esetén lassulás figyelhető meg:
- A memóriához kell nyúlni, ez pedig szűk keresztmetszetet jelent



Több szálú elérés

- Inkrementáláskor drasztikusan romlik a helyzet
- Fenn kell tartani a cache koherenciát, ez pedig időt vesz igénybe



CPUID

- CPUID: Az Intel processzoraiban elérhető utasítás, amely segítségével rengeteg CPU jellemzőt kiolvashatunk
- A CPUID előtt feltöltjük az eax, ebx, ecx, edx regisztereket a paraméterekkel
- Lefuttatjuk a CPUID-t
- A válasz az eax, ebx, ecx és edx regiszterekben lesz

Processzor nevének kiolvasása

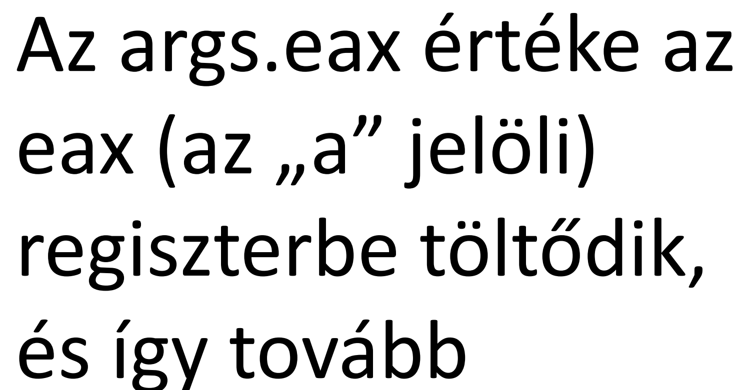
- Az eax-be töltsük be a 80000002h értéket, majd futtassuk a CPUID-t
- Ekkor a 4 regiszterbe betöltődik a név első 4*4 bájtja
- Folytassuk ezt a 80000003h és 80000004h értékekkel

CPUID meghívása gcc-vel

```
typedef struct Registers {
    unsigned int eax;
    unsigned int ebx;
    unsigned int ecx;
    unsigned int edx;
} Registers;


Registers cpuid(Registers args) {
    asm("cpuid"
    : "=a"(args.eax),
          "=b"(args.ebx),
          "=c"(args.ecx),
          "=d"(args.edx)
        : "a"(args.eax),
          "b"(args.ebx),
          "c"(args.ecx),
          "d"(args.edx)
        );
    return args;
}
```

CPUID meghívása gcc-vel

```
typedef struct Registers {
    unsigned int eax;
    unsigned int ebx;
    unsigned int ecx;
    unsigned int edx;
} Registers;

Registers cpuid(Registers args) {
    asm("cpuid"
    : "=a"(args.eax),
            "=b"(args.ebx),
            "=c"(args.ecx),
            "=d"(args.edx)
        : "a"(args.eax),
          "b"(args.ebx),
          "c"(args.ecx),
          "d"(args.edx)
        );
    return args;
}
```



CPUID meghívása gcc-vel

```
typedef struct Registers {
    unsigned int eax;
    unsigned int ebx;
    unsigned int ecx;
    unsigned int edx;
} Registers;

Registers cpuid(Registers args) {
    asm("cpuid"
    : "=a"(args.eax),
            "=b"(args.ebx),
            "=c"(args.ecx),
            "=d"(args.edx)
        : "a"(args.eax),
          "b"(args.ebx),
          "c"(args.ecx),
          "d"(args.edx)
        );
    return args;
}
```

Az args.eax értéke az eax (az „a” jelöli) regiszterből töltődik vissza, miután az assembly kód lefutott

Cache detektálás: struktúra

```
typedef enum CACHE_TYPE {
    DATA_CACHE = 1,
    INSTRUCTION_CACHE = 2,
    UNIFIED_CACHE = 3
} CACHE_TYPE;

typedef struct CacheInfo {
    int threads;
    int fullAssociative;
    int level;
    CACHE_TYPE type;
    int associativity;
    int linePartitions;
    int lineSize;
    int sets;
    int size;
} CacheInfo;
```

Cache detektálás: kód

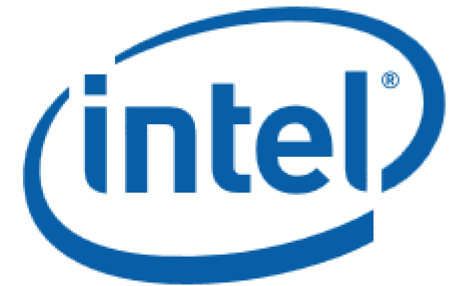
```
CacheInfo getCacheInfo(int cacheId) {
    Registers regs;
    regs.eax = 4;
    regs.ecx = cacheId;
    regs.ebx = regs.edx = 0;
    regs = cpuid(regs);

    CacheInfo info;
    info.threads = getBits(regs.eax, 0, 4) + 1;
    info.fullAssociative = getBits(regs.eax, 9, 9);
    info.level = getBits(regs.eax, 5, 7);
    info.type = getBits(regs.eax, 0, 4);
    info.associativity = getBits(regs.ebx, 22, 31) + 1;
    info.linePartitions = getBits(regs.ebx, 12, 21) + 1;
    info.lineSize = getBits(regs.ebx, 0, 11) + 1;
    info.sets = regs.ecx + 1;
    info.size = info.associativity *
            info.linePartitions *
            info.lineSize *
            info.sets;
    return info;
}
```

C:\Qt\Qt5.1.1\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
CPU name:           Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
Cache count: 4
Number of threads: 2
Full associative: 0
Level: 1
Data cache
Associativity: 8
Line partitions: 1
Line size: 64
Number of sets: 64
Size: 32K
----------------------

Number of threads: 3
Full associative: 0
Level: 1
Instruction cache
Associativity: 8
Line partitions: 1
Line size: 64
Number of sets: 64
Size: 32K
----------------------

Number of threads: 4
Full associative: 0
Level: 2
Unified cache
Associativity: 8
Line partitions: 1
Line size: 64
Number of sets: 512
Size: 256K
----------------------

Number of threads: 4
Full associative: 0
Level: 3
Unified cache
Associativity: 16
Line partitions: 1
Line size: 64
Number of sets: 8192
Size: 8192K
----------------------
```

Források

- Ulrich Drepper: What Every Programmer Should Know About Memory



Intel® Processor Identification and the CPUID Instruction

Application Note 485

August 2009

Köszönöm a figyelmet!